TU Clausthal

Clausthal University of Technology

Hochleistungsrechnen und Echtzeit in virtualisierten Maschinen und Clouds - Die Intel Virtualisierungshilfen

Harald Richter, Alexander Keidel

IfI Technical Report Series IfI-14-03

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Federico Schlesinger
Contact: federico.schlesinger@tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Inhaltsverzeichnis

Abbildungsverzeichnis

Hochleistungsrechnen und Echtzeit in virtualisierten Maschinen und Clouds - Die Intel Virtualisierungshilfen

Harald Richter, Alexander Keidel

Technische Informatik und Rechnersysteme
TU Clausthal
hri@tu-Clausthal.de

Zusammenfassung

Im Beitrag werden zuerst die Hardware- und Software-Voraussetzungen dargestellt, die erfüllt sein müssen, um aus einem Rechen-Cluster ein System zumachen, das entweder zum Hochleistungsrechnen befähigt ist, oder in weicher Echtzeit reagieren kann. Danach wird eine Übersicht über Methoden und Techniken der Ressourcen-Virtualisierung in einem Rechner gegeben. Es werden die Arbeitsweise und die Probleme von Hypervisoren und virtuellen Maschinen aufgezeigt. Darüberhinaus werden Informationen zu PCI Express präsentiert, die für das Verständnis der Intel-Virtualisierunghilfen notwendig sind. Im einzelnen werden die hardware-basierten Virtualisierungshilfen VT, VT-x, VT-c, VMX, VMCS, EPT, I/OAT, VMDq, SR-IOV, MR-IOV und VT-d besprochen. Anschließend werden der Mellanox Messaging Accelerator MXM und der Beschleuniger Open-MX von Myrinet vorgestellt und besprochen, die beide TCP/IP innerhalb eines Clusters optimieren. Außerdem werden Hinweise bzgl. der Kombinierbarkeit dieser Virtualisierungstechniken und hinsichtlich Ihrer Unterstützung durch KVM, Xen und OpenStack gegeben. Mit Hilfe der dargestellten Techniken kann man übliche Server, auf denen ein Hypervisor oder eine Cloud ausgeführt wird, zu einem virtualisierten HPC Cluster oder einem virtualisierten Echtzeitdatenerfassungs- und -verarbeitungs -system erweitern. Ebenso ist es möglich, eine Standard-Cloud zu einer HPC Cloud aufzuwerten, die parallele Codes effizient abarbeiten kann. Schließlich ist es möglich, eine Standard-Cloud mit der Fähigkeit zu weicher Echtzeit anzureichern. Für Anwender, die ältere Server länger betreiben wollen, werden Tuning-Tipps gegeben. [1]

[1] Der Beitrag wird erscheinen in: H. Richter, A. Keidel, Hochleistungsrechnen und Echtzeit in virtualisierten Maschinen und Clouds - Die Intel Virtualisierungshilfen, in IfI Technical Report Series ISSN 1860-8477, http://www.in.tu-clausthal.de/forschung/technical-reports/, editor: Department of Computer Science, Clausthal University of Technology, Germany, 2014.

1 Einleitung

Virtuelle Maschinen (VMs) in Rechenclustern und Clouds sind mit Hilfe eines Hypervisors bzw. eines Cloud-Betriebssystems gut skalierbar und leicht administrierbar, aber sie sind leider nicht immer effizient, was das parallele Rechnen anbetrifft und für Echtzeit zudem nicht verwendbar, weil Echtzeit eine schnelle und deterministische Kommunikation zwischen VMs untereinander und mit der Peripherie erfordert. Intel, AMD und Hersteller von PCIe-basierten Endgeräte -insbesonders von Netzwerkkarten- bieten allerdings seit einigen Jahren hardware-basierte Virtualisierungsunterstützungen für Hypervisoren und Clouds an, die diese Ineffizienzen abmildern oder sogar ganz beseitigen. Diese Beschleuniger sind jedoch in Funktion und Konfiguration komplex, so dass Server, virtualisierte Infrastrukturen und damit auch Clouds des öfteren unter ihren Möglichkeiten betrieben werden.
Das Ziel dieses Beitrags ist es, zu beschreiben, welche Hardware-basierten Unterstützungen es für die Virtualisierung des Rechnens und der Kommunikation gibt, was die dahinterliegenden Konzepte und Methoden sind, und wie man diese so nützt, dass Server mit höherer Effizienz Hypervisoren und Clouds ausführen können. Dazu wird die Virtualisierungstechnologie „Intel VT“ bestehend aus VT-x, VT-c und VT-d beschrieben, zusammen mit der „Intel I/O Acceleration Technology I/OAT“. Darüberhinaus werden zwei Netzwerktechniken von den Firmen Mellanox und Myrinet dargestellt, die die Ausführungsgeschwindigkeit des TCP/IP Stacks innerhalb eines Rechen-Clusters erheblich beschleunigen.
Der Beitrag richtet sich an Rechenzentren mit Interesse an Hochleistungsrechnen (High Performance Computing, HPC) auf einem Server Cluster oder in einer Cloud. Er richtet sich aber auch an Institute, die eine Echtzeit- Datenerfassung mit Hilfe von virtuellen Maschinen oder sogar in einer Cloud durchführen wollen. Obwohl Echtzeit und HPC verschiedene Anwendungsfelder darstellen, versuchen beide, die Effizienz der Virtualisierung zu maximieren und die Latenz der Interprozesskommunikation deterministisch und minimal zu gestalten. In diesem Sinne sind Echtzeit und HPC die beiden Seiten derselben Medaille. Der Beitrag richtet sich schließlich auch an Rechenzentren, die ältere Server länger betreiben wollen, indem sie deren Leistung durch Tuning steigern. Die im Beitrag beschriebenen Konzepte und Methoden werden exemplarisch für die Hypervisoren KVM und XEN und für die weit verbreitete „OpenStack“ Cloud Middleware dargestellt. Als Server- Prozessoren werden Intel Pentium Multicore-CPUs bzw. dazu verwandte Prozessoren von AMD vorausgesetzt.

2 Voraussetzungen für HPC, Echtzeit und Server-Tuning

Damit es überhaupt möglich ist, VMs und Clouds für HPC und Echtzeit einzusetzen, müssen bestimmte Mindestvoraussetzungen sowohl auf der Hardware- als auf der Software-Seite gegeben sein. Diese werden nachfolgend dargestellt.

2.1 Hardware-Voraussetzungen

Als erstes muss geprüft gewerden, ob die zu tunende Server und von ihrer Hardware-Ausstattung her für HPC bzw. Echtzeit geeignet sind. Dies ist der Fall, wenn folgende Voraussetzungen erfüllt sind:

1. Die CPUs der Server, die für HPC gedacht sind, müssen die „Intel VT-x-“ und „VT-c“-Virtualisierungshilfen aufweisen, zumindest in einer frühen Version. Diese Hilfen unterstützen und beschleunigen Hypervisoren, VMs und Clouds hardware-mäßig und werden in den nachfolgenden Kapiteln beschrieben. Die Virtualisierungshilfe Intel VT-d, die ebenfalls weiter unten dargestellt wird, ist unseres Erachtens nach nur für Anwendungen aus dem Echtzeitbereich relevant, da dabei ein peripheres PCI Express-Gerät fest einer VM zuordnet wird, so dass deterministische Zeitverhältnisse beim Zugriff auf das Gerät entstehen.

2. Weiterhin müssen in den Servern des Clusters oder der Cloud mindestens 16 GB RAM an realem Hauptspeicher pro CPU installiert sein. Dabei darf der reale Speicher nicht durch Benutzeranwendungen oder durch eine zu hohe Zahl von VMs überbucht (over-committed) werden.

3. Darüberhinaus ist es notwendig, dass schnelle Netzwerkverbindungen mit einer Latenz im einstelligen Mikrosekunden-Bereich vorhanden sind, wie es ab 10 Gbit/s-Ethernet, -Infiniband oder -Myrinet der Fall ist. Speziell im HPC-Bereich werden traditionell Infiniband-Netzwerkkarte zusammen mit einem Infiniband-Switch eingesetzt, der die Server zu einem Cluster oder einer Cloud koppelt. Aktuelle Switches und Netzwerkkarten arbeiten bei Kupferkabeln mit Geschwindigkeiten bis 56 Gbit/s, sind also fünf Mal schneller als die klassische 10 Gbit/s-Cluster-Kopplung für HPC.

4. Schließlich ist es notwendig, die richtige Kombination von Beschleunigern einzusetzen, so dass sich diese nicht gegenseitig behindern oder ausschließen, wie es z.B. bei SR-IOV und VMDq der Fall ist. Einige Beschleuniger wie die Extended Page Tables (EPT) sind außerdem stark von

der jeweiligen Benutzeranwendung abhängig und können in bestimmten Situationen sogar zu Leistungseinbußen anstelle einer Beschleunigung führen.

Ob VT-x, VT-c oder VT-d in einer CPU, einem Motherboard oder einem PCIe-Gerät vorhanden sind, kann über das BIOS und anhand der Dokumentation herausgefunden werden. Bei Servern, die nach dem Jahr 2006 gebaut wurden, ist das in einer mehr oder weniger guten Ausbaustufe immer gegeben, sofern sie CPUs, Chipsätze und NICs von Intel oder AMD enthalten. In solchen Fällen, kann man auf einer einzelnen VM fast so schnell, wie auf einer echten Maschine rechnen. Damit ist allerdings noch nichts über die Effizienz bei der Inter-VM-Kommunikation ausgesagt, die bei parallelen HPC-Anwendungen benötigt wird, und auch nichts über die Effizienz des Datentransfers zwischen VM und virtueller Peripherie. Nur beides zusammen, Hardware-Beschleunigung beim Rechnen und beim Kommunizieren erlauben Hochleistungsrechnen oder Echtzeit auf virtualisierten Rechnerressourcen oder in einer Cloud.

2.2 Software-Voraussetzungen

Um das Potential, das VMs und Clouds haben, voll auszuschöpfen, ist es erforderlich, bestimmte BIOS-, Hypervisor- und Cloud Middleware-Optionen freizuschalten bzw. richtig zu konfigurieren und ggf. einige Zusatzprogramme zu installieren. Diese werden in den nachfolgenden Kapiteln beschrieben. Schließlich ist es bei den HPC-Anwendungen, die das weit verbreitete Message Passing Interface (MPI) benützen, notwendig, dass die MPI- Implementierung direkt auf der Network Interface Card (NIC) im Falle von Ethernet oder dem Host Channel Adapter (HCA) im Falle von Infiniband aufsitzt und nicht TCP/IP verwendet, zumindest nicht in seiner Vollversion, da TCP/IP die Interprozessorkommunikation drastisch verlangsamt.
Hinweis: NICs werden in der Infiniband-Termininologie meist in HCA (Host Channel Adapter und TCA (Target Channel Adapter) aufgeteilt. Ein HCA ist für den gesamten Rechner zuständig und damit mit einer NIC vergleichbar. Mit einem TCA wiederum kann nur ein einzelnes Peripheriegerät, wie z.B eine einzelne Festplatte, an das Infiniband-Netz angeschlossen werden. Siehe dazu auch http://en.wikipedia.org/wiki/InfiniBand
Im vorliegenden Report werden zwei SpezialImplementierungen von TCP/IP-beschrieben, die es erlauben, MPI zusammen mit TCP/IP zumindest innerhalb einer Cloud für Punkt-zu-Punkt-Verbindungen zu betreiben, wenn auch nicht zusammen mit dem Internet, da das weltweite Routing fehlt. Das bedeutet: über das Internet verteilte Anwendungen und HPC bzw. Echtzeit schließen sich gegenseitig aus.
Für den Fall, dass ein Server oder eine Cloud als Echtzeit-Datenerfassungs- oder -verarbeitungssystem eingesetzt werden soll, müssen die sog. System

Management Interrupts (SMIs), die in IA32- und IA32/64-Architekturen standardmäßig verwendet werden, abgeschaltet oder umgangen werden, Ein SMI schaltet die CPU in den hochpriviligierten System Management Mode (SMM), in dem sowohl das Betriebssystem als auch übliche Anwendungen auf unbestimmte Zeit unterbrochen werden. In diesem Modus kann auch auf externe Interrupts nicht mehr reagiert werden. Latenzen bis 100 ms sind so möglich, und sie sind nur schwer bis gar nicht vorhersagbar. SMIs erzeugen für Echtzeitanwendungen dieselben Probleme wie der Garbage Collection- Speicherbereinigungs -mechanismus von Java und sind somit meistens nicht tolerierbar.
Des Weiteren müssen Energiespar-Optionen wie z.B. ACPI abgeschaltet werden, da sie ebenfalls unvorhersagbare Latenzen erzeugen können.
Hinweis: Siehe dazu
https://rt.wiki.kernel.org/index.php/HOWTO:_Build_an_RT-application
Darüberhinaus muss vom Systemadministrator sichergestellt werden, dass die VMs auf einem Server nie mehr an virtuellem Speicher allozieren als real vorhanden ist. Ein Überbuchen (Over-committing) des realen Hauptspeichers ist nicht erlaubt, andernfalls wird das paging des Host-Betriebssystems aktiviert, und es erfolgt ein ununterbrochenes Seitentauschen zwischen der Festplatte und dem Hauptspeicher. Sobald paging einsetzt, wird HPC extrem ineffizient, und das Echtzeitverhalten wird indeterministisch, da zu beliebigen Zeitpunkten mit mehreren Dutzend Millisekunden Zeitverlust aufgrund von Plattenzugriffen gerechnet werden muss, die für den Anwender nicht vorhersagbar sind. Schließlich muss der Benutzer einer VM dafür sorgen, dass auch das Guest-Betriebssystem und die Benutzeranwendung in jeder VM nicht in den Paging-Modus bei der virtuellen Festplatte gerät. Dies kann er dadurch ereichen, dass er die Zahl der Gitterpunkte und Zeitschritte seiner parallelen Anwendung, die z.B. eine Simulation sein kann, so weit absenkt, dass seine Anwendung komplett in den realen Hauptspeicher passt, der seiner VM bei ihrem Start zugeteilt wurde.

3 Übersicht über die Intel- Virtualisierungstechniken

Intel und AMD bieten seit einigen Jahren in ihren Prozessoren, Motherboard-Chip-Sätzen und Netzwerkkarten diverse Hardware-basierte Virtualisierungshilfen für das Rechnen auf einer VM und für das Kommunizieren zwischen VMs und mit der Peripherie an. Diese Hilfen wurden und werden fortlaufend verbessert und firmieren mittlerweile kollektiv unter der Bezeichnung **Intel Virtualization Technology VT**. Gemäß [1] und [2] besteht Intel VT aus den drei einzelnen Technologiepaketen **Intel VT-x, VT-c** und **VT-d**. VT-x [3], [4] steht für „Intel Virtualization Technology for x86-Architectures“

und beinhaltet diverse Hardware-Beschleuniger für Prozessoren mit IA32- und IA32/64-Architektur. Bei den IA64-Architekturen, d.h. bei den Itanium-Prozessoren, werden diese Beschleuniger unter dem Oberbegriff VT-i zusammengefasst („i“ wie Itanium).
Intel VT-c [6] steht für „Intel Virtualization Technology for Connectivity“ und ist ein Bündel von Hardware-basierten Beschleunigern für Intel- Netzwerkkarten und für die VM-Kommunikation insgesamt. Intel VT-d schließlich ist die Abkürzung für „Virtualization Technology for Directed I/O“ und bezeichnet eine einzelne Beschleunigungshilfe, die im Intel Chipsatz integriert war. Der Chipsatz bestand früher aus zwei Komponenten, der Northbridge und der Southbridge. Mittlerweile ist jedoch die Northbridge mit der CPU verschmolzen, so dass der „Chipsatz“ nur noch aus der Southbridge besteht, die von Intel seitdem als I/O Controller Hub (ICH) bezeichnet wird. Die Intel VT-d-Technologie beinhaltet neben den Erweiterungen in der Southbridge auch Änderungen in der CPU, so dass es dadurch eine gewisse Überlappung mit VT-x gibt.

3.1 Intel VT-x und VT-c

3.1.1 VT-x

VT-x ist der Oberbegriff von Intel für alle Maßnahmen zum effizienten sequentiellen Rechnen auf einer einzelnen VM. Die Recheneffizienz beruht darauf das der Verwaltungszusatzaufwand, der aufgrund der Virtualisierung der Server-Hardware entsteht, durch neue Hardware im Server abgesenkt wird, die nicht virtualisiert wird. Intel VT-x besteht aus folgenden einzelnen Technologien:

1. Der Virtual Machine Extension (**VMX**) im Befehlssatz der CPU
2. Der Virtual Maschine Control Structure (**VMCS**) im Hauptspeicher
3. Den Extended Page Tables (**EPT**), die eine Erweiterung im Hauptspeicher aber in der CPU darstellen

3.1.2 VT-c

VT-c ist der Oberbegriff von Intel für alle Maßnahmen zum effizienten Kommunizieren zwischen VMs untereinander und mit der Peripherie. Insgesamt beinhaltet VT-c derzeit die folgenden Technologien:

1. Die „I/O Acceleration Technology“ (**I/OAT**). Durch I/OAT wird die ganze Kette der an I/O beteiligten Rechnerkomponenten beschleunigt. Die I/O Acceleration Technology besteht ihrerseits aus einem weiteren Bündel von Hardware-Beschleunigern für reale Kommunikation über reale

Peripherie. Als Konsequenz werden z.B. auch die Ausführungszeiten von TCP/IP oder von MPI über Infiniband verkürzt, und Netzwerkkarten können bis an deren physikalischen Bandbreitegrenzen betrieben werden. Die I/O-Virtualisierung profitiert indirekt von I/OAT über eine schnellere Abwicklung der realen I/O.

2. Die „Single Root Virtualization“ (**SR-IOV**). SR-IOV beruht auf einer Virtualisierungsunterstützung mit Hilfe der PCI-Express-Schnittstellen auf dem Motherboard und in den Netzwerkkarten. Durch SR-IOV können periphere Geräte, die an PCI Express (PCIe) angeschlossen sind, im Multiplexbetrieb von verschiedenen VMs gleichzeitig genutzt werden. SR-IOV wird von Intel manchmal auch als „Virtual Machine Direct Connect“ (**VMDC**) bezeichnet, obwohl diese Technologie gar nicht von Intel stammt, sondern von der PCIe Special Interest Group (PCIe SIG). Eine Variante von SR-IOV ist die „Multi Root Virtualization“ (**MR-IOV**), die nur für sog. Blade Server relevant ist. Unter Blade Server versteht man Rechner, die einzelne Baugruppen wie Netzteile, Lüfter und Peripherie nicht mehr individuell enthalten, sondern gruppenweise nutzen. Blade Server können so kompakter aufgebaut werden als normale Server. Da sie auch Netzwerkkarten gemeinsam nutzen, ist SR-I/O nicht anwendbar, weshalb MR-IOV entwickelt wurde.

3. Die „Virtual Machine Device Queues“ (**VMDq**). Damit wird ebenfalls der Multiplexbetrieb peripherer Geräte unterstützt. SR-IOV und VMDq schließen sich gegenseitig aus.

4. Die „Virtualization Technology for Direct I/O“ (**VT-d**). Diese Technik erlaubt es, dass ein reales peripheres Gerät statisch einer VM zugeordnet wird, sodass die VM direkt darauf zugreifen kann, ohne über die Zwischenstufen von Hypervisor und virtueller Peripherie gehen zu müssen.

Alle diese Virtualisierungshilfen bewirken, dass der Hypervisor und damit der Server entlastet und wesentliche Teile der Aufgaben auf spezielle Hardware-Komponenten in der CPU, der Southbridge und der Peripherie ausgelagert werden.
Im weiteren Verlauf des Reports werden als Peripherie überwiegend PCIe-basierte Ethernet-Karten betrachtet. Die dabei gemachten Aussagen gelten aber auch für Infiniband NICs oder für Myrinet-Karten.

3.2 Aktivieren von Intel VT-x und VT-c bzw. von AMD V und Vi

Jede Virtualisierungshilfe aus VT-x/VT-c bzw. AMD V/Vi muss im BIOS der Server einzeln aktiviert werden. Beispielsweise muss Intel EPT bzw. AMD

RVI im Bios eingeschaltet sein, um es nutzen zu können. Informationen zur Aktivierung findet man in der jeweiligen BIOS-Dokumentation der Server, sowie in den nachfolgenden Kapiteln, in denen auch die Wirkungsweise und Vorteile dieser Maßnahmen beschrieben sind. Darüberhinaus finden Sie dazu Infos unter [4] und [22].

4 Übersicht über TCP/IP-Beschleuniger im Cluster

Zur Verbesserung der Performance für High-Performance Computing und für Echtzeit besteht ein erster Ansatz darin, TCP überall dort nicht zu installieren, wo es nicht gebraucht wird. Dies führt zu einem deutlich geringeren Overhead. Da TCP sowohl im Host- als auch im Guest-Betriebssystem, d.h. insgesamt zwei Mal pro Sender und zwei Mal pro Empfänger ausgeführt wird, trägt eine sinnvolle Teilinstallation von TCP darüberhinaus auch zu einer Energieeinsparung im Cluster und in der Cloud bei, da die Cores von TCP-Sender und Empfänger weniger beschäftigt sind. Dasselbe gilt für IP.
Ein zweiter Ansatz zur Verbesserung der Performance für High-Performance Computing und für Echtzeit besteht darin, TCP/UDP dort zu tunen, wo es tatsächlich benötigt wird. Dieser Ansatz wird von uns favorisiert. Im wesentlichen gibt es zwei TCP/IP-Tuning-Möglichkeiten:

1. Durch die Verwendung von Standard-TCP/UDP zusammen mit der Intel I/O Acceleration Technology I/OAT

2. Durch die Installation einer Spezialversion von TCP und UDP, die Schnittstellenkompatibel zur Originalversion ist, d.h. die Berkeley Sockets unterstützt. Beispiele dafür sind das Messaging Accelerator-Protokoll MXM von der Fa. Mellanox oder Open-MX von Myrinet.

Diese Maßnahme bewirkt von allen in diesem Report beschriebenen Beschleunigern und Virtualisierungshilfen den größten Gewinn für TCP/IP-basierte Inter-VM-Kommunikation. Durch das Messaging Accelerator-Protokoll von Mellanox wird laut Herstellerangaben die Latenz bei der Datenübertragung von Anwenderprozess zu Anwenderprozess, die über ein 10G Ethernet von Mellanox gekoppelt sind, auf 1,6 ms verringert, bei UDP sind es sogar 1,3 ms. Diese Zahlen werden nur noch von den Echtzeit-Rechnernetzen CarRing 3 (CR3) [5] und CarRing 4 (CR4) übertroffen, die am Lehrstuhl entwickelt wurden bzw. werden.

5 Technische Grundlagen der Virtualisierungst- echniken

Um die Funktionsweise der Virtualisierungshilfen verstehen zu können, müssen zuerst die Grundlagen dafür erläutert werden. Nachfolgend wird dazu ein Überblick über Virtualisierung, Hypervisoren, virtuellen Maschinen und Clouds gegeben und deren wichtigsten Eigenschaften und Arbeitsweisen dargestellt. Die konkreten Virtualisierungshilfen von Intel/AMD kann man daraus ableiten.

5.1 Clouds

Eine Cloud ist, einfach gesprochen, eine Sammlung von Management- Diensten, um damit virtuelle Maschinen in einem Cluster zu erzeugen, zu betreiben und zu verwalten.
Hinweis: Als umfassende Definition einer Cloud hat sich mittlerweile in der Literatur, die Definition des U.S. National Institute of Standards (NIST) durchgesetzt.
Siehe: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.
Dies erfolgt in enger Zusammenarbeit den Hypervisoren auf den einzelnen Rechnern, aus denen das Cluster besteht. Die Cloud ist dabei oberhalb der Hypervisoren angesiedelt. Eine Software zur Realisierung einer Cloud wird häufig auch als Cloud OS bezeichnet, da ein Cloud OS, analog zu einem traditionellen Betriebssystem für reale Hardware, ebenfalls einfach zu benutzende Schnittstellen bereitstellt, diesmal allerdings für virtuelle Ressourcen. Typische Eigenschaften, die mit den virtuellen Ressourcen einer Cloud erreicht werden können, sind das sog. Resource Pooling, das zu einem Lastausgleich zwischen realen Rechnern führt, und die sog. Rapid Elasticity, die es erlaubt, bei Bedarf schnell neue virtuelle Maschinen zu starten.
Dafür stellt die Cloud Cluster-weit zahlreiche Dienste zur Verfügung. Bei der Icehouse-Version von OpenStack bespielsweise sind das ein Metascheduler namens „Nova“, ein Netzwerkdienst namens „Neutron“ für die Verwaltung virtueller NICs, ein Identitäts- und Zugriffsrechte-Verwaltungssystem namens „Keystone“, eine web-basierte Benutzoberfläche namens „Horizon“, machmal auch „Dashboard“ genannt, einem Netzwerkdienst namens „Neutron“, der ein Software Defined Networking (SDN) virtueller NICs und Switches erlaubt, sowie zwei Speicherdienste namens „Cinder“, „Swift“ und ein Leistungsmessungsdienst namens „Telemetry“, um nur die wichtigsten zu nennen. Für das Cluster-weite Management virtueller Ressourcen existiert bei OpenStack eine REST-basierte API.
Hinweis: REST ist eine Implementierung des http-Protokolls, bei der auch die Methode DELETE funktioniert, sowie ein bestimmter Satz von Regeln („Architekturstil“), die es u.a. erlauben, einen statuslosen Web Service mit Hilfe

von http-Methoden und sog. Uniform Resource Identifiern (URIs) zu realisieren.
Eine Besonderheit bei OpenStack ist, dass die Nutzung einiger Dienste der Cloud über die REST API nicht möglich ist, sondern nur deren Verwaltung. Der Zugriff auf den Datenbankdienst beispielsweise erfolgt von einer Benutzanwendung aus über SQL-Aufrufe, der Zugriff auf den blockbasierten Speicherdienst Cinder über readf/printf. Dies stellt die Kompatibilität der Cloud mit existierenden Standards sicher.

5.2 Intel Zwiebelschalen-Programmiermodell

Das Zwiebelschalen-Programmiermodell für IA32- und x86/64-Software ist wichtig für das Verständnis, wie Hypervisoren und Virtualisierungshilfen funktionieren. Das Modell besteht aus mehreren Schichten, die als Ringe bezeichnet werden. Im ursprünglichen Zwiebelschalenmodell der IA32- Software wurde das Host OS in der innersten Schale -dem Ring 0- ausgeführt und besass alle Rechte für den Zugriff auf die Hardware. Anwendungsprogramme liefen in Ring 3 und hatten einen deutlich eingeschränkteren Zugriff. Dieses Modell hatte aber das Problem, dass es nicht für den gleichzeitigen Betrieb mehrerer Betriebssysteme, d.h. für ein Host OS mit multiplen Guest OS (VMs) in einem Rechner ausgelegt war.
Hinweis: die Begriffe Guest OS und Host OS werden nachfolgend erläutert.
Vielmehr konnte stets nur ein Betriebssystem, in diesem Fall das Host OS, in Ring 0 laufen. Kein Guest OS hatte den Zugriff auf Festplatte oder sonstige Peripherie, denn das Guest OS konnte nur im Ring 3, d.h. als Benutzeranwendung betrieben werden. Ein Host OS mit parallelem Guest OS waren somit nicht möglich, genauso wenig wie mehrere VMs, die gleichzeitig auf dasselbe reale Endgerät zugreifen können.

5.2.1 Funktionsweise des Zwiebelschalenmodell mit Hypervisor

Um dem Hypervisor die Arbeit zu erleichtern, wurden von Intel im Zwiebelschalen-modell die einzelnen GuestOS nach Ring 1 verschoben, während in Ring 0 nur das Host OS zusammen mit dem Hypervisor wie z.B. „KVM“ lief.
Hinweis: Bei einzelnen Hypervisoren, die über eigene Gerätetreiber verfügen, wie z.B. XEN, läuft sogar nur der Hypervisor in Ring 0.
Das Verschieben der Guest OS nach Ring 1 wurde auch als „Ring De-Privileging“ bezeichnet. Die Folge von Ring De-Privileging ist, dass dann, wenn ein GuestOS IA32- oder IA32-64-Befehle mit Ring 0-Privilegien ausführen möchte -wie z.B. das Löschen eines Daten-Caches oder das Schreiben in ein peripheres Gerät- wird ein CPU-Zugriffsrechtefehler (Access Privilege Violation) erzeugt. Diese CPU Execption (Trap) wird vom Hypervisor abgefangen. Anschließend emuliert der Hypervisor das Verhalten des in Ring 1 nicht erlaubten Ring 0-Befehls des Guest OS, ähnlich wie ein Debugger das Verhalten

des zu debuggenden Programms nachahmt. Ohne das Abfangen des Guest OS-Ring 0-Befehls durch den Hypervisor hätte das Guest OS die Kontrolle über die reale Maschine, was bei der Virtualisierung verboten ist.
Guest OS Ring 0-Befehle heißen „Root-Instruktionen“, während alle anderen Befehle „Non-Root-Instruktionen“ sind. Root-Befehle laufen im priviligierten Kernel Mode der CPU.
Das Abfangen von Root-Instruktionen bringt allerdings einen hohen Software-Verwaltungsaufwand mit sich und war ein Grund für den Beginn von Intel VT-x im Vanderpool Chipsatz im Jahre 2006. Non-Root-Instruktionen hingegen erfordern kein Abfangen durch den Hypervisor, da ihre Ausführung unkritisch ist.

5.2.2 Privilegien im Zwiebelschalenmodell

Im Modell hat der innerste Ring die höchsten Privilegien und der äußerste die niedrigsten. Im innersten Ring befindet sich das Host OS-sofern vorhanden- und der Hypervisor. In den mittleren Ringen sind die virtuellen Maschinen (VMs) mit Guest OS und darüber die Cloud Middleware angesiedelt. Ganz aussen sind die Benutzeranwendungen lokalisiert.

5.3 Hypervisoren

Ein Hypervisor ist eine Software-Schicht, die zwischen dem Host OS und den VMs sitzt. Hypervisoren erlauben die Virtualisierung aller Hardware-Komponenten eines Rechners, indem Sie deren Funktion nachahmen. Man unterscheidet zwischen Typ 1- und Typ 2-Hypervisor. Ein Typ1-Hypervisor, wie z.B. XEN, sitzt direkt auf der realen Maschine auf und hat eigene Gerätetreiber. Er stellt somit ein Spezialbetriebssystem „im Kleinen“ dar. Ein Typ 2-Hypervisor, wie z.B. KVM, hat von Haus aus keine Gerätetreiber, weil er innerhalb eines Standard-Betriebssystem läuft. Die neueren Version von KVM haben allerdings einige Gerätetreiber in einer Bibliothek namens libvirt, die einen effizienten Zugriff des Guest OS auf die Peripherie durch die sog. „Paravirtualisierung“ erlauben, die später beschrieben wird.
Jeder Hypervisor ist mit Betriebssystem-Privilegien ausgestattet, d.h., er kann alle CPU-Befehle ausführen und auf alle Daten, Hauptspeicherbereiche und Geräte zugreifen. Beide Typen von Hypervisoren (Typ 1 und Typ 2) erlauben das Starten, Verwalten, Stoppen und Beenden von mehreren virtuellen Maschinen auf demselben realen Rechner. In jeder VM läuft i.d.R. ein Betriebssystem, das sog. Guest OS. Pro realer Maschine gibt es höchstens ein Host OS und maximal so viele Guest OS, wie VMs existieren.
Meist existiert ein Hypervisor pro Host OS, der sich um alle VMs auf einer realen Maschine kümmert. Eine Variante des Hypervisors ist der sog. Virtual Machine Manager (**VMM**), wie er z.B. in vSphere von VMware eingesetzt wird. Im Gegensatz zum Hypervisor gibt es so viele VMMs in einem

Host OS, wie es VMs gibt, denn je ein VMM ist einer VM zugeordnet. Wenn in diesem Dokument von „Hypervisor“ die Rede ist, dann ist damit auch die VMM-Variante gemeint.

5.4 Host OS und Guest OS

Das Basisbetriebssystem, in dem der Hypervisor, sowie -sofern vorhanden- auch das Cloud-Betriebssystem läuft, wird als Host OS bezeichnet. In der Regel wird als Host OS ein Standardbetriebssystem wie Linux oder Windows verwendet. Auf jeder virtuellen Maschine (VM) kann seinerseits ein Guest OS ausgeführt werden, das z.B. ebenfalls Windows oder Linux sein kann.

5.5 Virtualisierung

Virtualisierung ist das Schaffen von virtuellen Maschinen, die sich die realen Ressourcen des Servers bestehend aus CPUs, Cores, Hauptspeicher, Peripherie und Massenspeicher im Time Sharing-Betrieb teilen. Ein Spezialfall der Virtualisierung stellt die I/O-Virtualisierung dar. Jede virtuelle Peripherie agiert aus VM-Sicht unabhängig von Peripherie in anderen VMs, genauso wie die VMs untereinander getrennt sind. VMs auf demselben realen Rechner kommunizieren so miteinander, als ob sie in verschiedenen realen Rechnern angesiedelt wären. Oft wird dazu TCP/IP zusammen mit 1G Ethernet-NICs verwendet, was für HPC und Echtzeit nicht relevant ist, da TCP/IP zu indeterministisch in Bandbreite und Latenz ist, und 1G Ethernet zu langsam ist.
Hypervisoren vollziehen die Virtualisierung der realen Rechnerkomponenten entweder nur in Software oder mit mehr oder weniger großer Unterstützung durch Hardware-/Firmware-Hilfen.
Insgesamt kann die Virtualisierung von Rechnerressourcen durch **Software-Simulation** der Hardware, durch sog. **Binary Translation**, durch **Paravirtualisierung** oder durch **Hardware-Virtualisierung** erfolgen. Möglich ist auch eine Kombination aus allen drei Virtualisierungstechniken, wie z.B eine Hardware-Virtualisierung der CPU und eine Paravirtualisierung der NIC. Alle Techniken werden im folgenden erläutert.

5.5.1 Software-Simulation der Hardware

Bei der Software-Simulation der Hardware werden die Hardware-Funktionen eines Rechners per Software in einer Simulation nachgebildet. Dies kann bis hinunter auf die Ebene einzelner CPU-Takte erfolgen, was aber sehr zeitraubend ist. Im Falle einer reinen Software-Simulation sind erhebliche Einbußen in der Leistungsfähigkeit der virtuellen Maschinen im Bereich von bis zu Faktor 10^5 in Kauf zu nehmen. Die Software-Simulation der Hardware ist für

unsere Zwecke nicht relevant.

5.5.2 Binary Translation des Guest OS und der Anwendung

Bei der Binary Translation wird das Guest OS und der ausführbare Benutzer-Code zur Laufzeit vom Hypervisor interpretiert und daraufhin untersucht, welche Betriebssystemaufrufe beide in einer VM absetzen. Diese Aufrufe werden on-the-fly vom Hypervisor abgefangen und durch eine Reihe von statements ersetzt, die bzgl. der VMs und deren Gastbetriebssysteme „sicher“ sind. Sicher heißt, dass zwischen VMs und innerhalb einer VM der Speicherschutz zwischen Prozessen sichergestellt ist, und dass keine Anwendung und kein Gastbetriebssystem die hohe Priorität des Host-Betriebssystems erlangt, obwohl dies zur Ausführung von Betriebssystemaufrufen eigentlich notwendig ist. Diese Aufgaben werden vom Hypervisor erledigt, der im Host-Betriebssystem läuft und anstelle des Guest OS und der Benutzeranwendung die erforderlichen Betriebssystemaufrufe selber durchführt und die Ergebnisse an das Guest OS und die Anwendung weiterreicht. Binary Translation ist für unsere Belange zu langsam und stellt auch eine veraltete Technik dar.

5.5.3 Paravirtualisierung

Bei der Paravirtualisierung werden im Guest OS neue Betriebssystemaufrufe in Form von speziell angepassten Gerätetreibern bereitstellt, die im großen Ganzen ähnlich zu denen sind, die im Host OS real existieren und tatsächlich ausgeführt werden können. D.h., bei der Paravirtualisierung ist eine Veränderung des Source Codes des Guest OS erforderlich. Indem es seinen eigenen Code analysisiert, ist sich das Guest OS auch potentiell "bewusst", dass es virtualisiert wurde.
Der Grund für das Schaffen neuer Betriebssystemaufrufe liegt darin, dass das Guest OS deswegen seine originalen Betriebssystemaufrufe nicht verwenden kann, weil es damit die Gastbetriebssysteme anderer VMs auf dem gleichen Server sowie das Host OS stören würde. Paravirtualisierung scheitert, wenn der Source Code wie z.B. bei Windows nicht zur Verfügung steht und der Hersteller des OS keine für Paravirtualizierung angepassten Treiber bereitstellt. Wenn Paravirtualisierung möglich ist, dann bietet sie jedoch eine rel. hohe Performanz, da die VM und der Hypervisor besonders einfach implementiert werden können und den geringsten Software-Overhead haben, da durch die angepassten Gerätetreiber alles nahtlos zusammenpasst. Ein Beispiel für Paravirtualisierung ist XEN, das für verschiedene Gastbetriebssyteme die sog. VMI-Schnittstelle für neue Betriebssystemaufrufe zur Verfügung stellt. Ein anderes Beispiel ist libvirt von KVM.

5.5.4 Hardware-Virtualisierung

Beispiele für Hardware-Virtualisierung sind die neueren Produkte von VMware oder KVM. VMWare begann ursprünglich mit der Binary Translation- Virtualisierung und setzt heute die Hardware-Virtualisierung ein. Xen hingegen begann mit Paravirtualisierung als Alternative zu VMWare, weil es die einfacher zu implementierende Technik war, und blieb dabei.
Bei der Hardware-Virtualisierung geht es im Gegensatz zur Paravirtualisierung darum, alle Guest OS-Betriebssystemaufrufe unverändert zu lassen. Zugriffe auf die reale Hardware werden über Exceptions (Traps) abgefangen, so dass das Guest OS nie die Kontrolle über die reale Maschine hat. Die vom Guest OS beabsichtigte Funktion wird statt dessen vom Hypervisor mit Hilfe des Host OS ausgeführt. Dem Guest OS wird vorgespielt, es habe z.B. einen Gerätetreiber aufgerufen, oder es habe die Page Tables konsultiert, oder es habe den TLB aktualisiert, oder es habe einen DMA-Transfer mit Interrupt durchgeführt. Dies ist unter Zuhilfenahme von Hardware-Virtualisierung mehr oder weniger effizient möglich. Die Hardware-Virtualisierung ist aufgrund der zunehmend besseren Unterstützung durch die Hersteller von CPUs, Motherboards und PCIe-Geräten das Mittel der Wahl geworden und entspricht dem Stand der Technik. Im weiteren Verlauf wird deshalb nur noch diese Art der Virtualisierung diskutiert.

5.6 Virtuelle Maschinen

Eine virtuelle Maschine ist eine Sammlung virtueller Ressourcen, die es erlauben, ein Guest OS und Benutzeranwendungen wie auf einer realen Maschine auszuführen. Typischerweise sind dazu virtualisierte CPUs mit virtuellen Cores, ein virtueller Hauptspeicher mit Peripherie, sowie virtueller Massenspeicher notwendig. Virtuelle Maschinen wurden u.a. aus Gründen besserer Serverauslastung und höherer Energieeffizienz eingeführt. Das Stichwort hierbei ist die sog. „Server Consolidation“, was als „Server-Zusammenführung“ übersetzt wird. Gemeint ist damit, dass schlecht ausgelastete Server ihre VMs auf besser ausgelastete Server verschieben können, um diese noch besser auszulasten. Anschließend gehen die ersteren in den Power Down-Modus und sparen Energie und Administrator-Ressourcen.
Eine VM besteht aus einer oder mehreren virtuellen CPUs, die in OpenStack als **vCPUs** bezeichnet werden, sowie aus virtuellem Hauptspeicher, inkl. einer virtuellen Speicherverwaltungseinheit (MMU).
Hinweis: Ein physikalischer Server verfügt heutzutage über zwei oder mehr Sockel für dementsprechend viele reale CPUs. Jede reale CPU wiederum besteht aus 4-16 Cores. Eine VM hingegen besteht bei OpenStack und anderen Cloud OS nur aus einer oder mehreren vCPUs, und jede vCPU besitzt genau einen virtuellen Core. Virtuelle Multicore-Architekturen werden dadurch realisiert, dass der VM einfach zusätzliche vCPUs hinzufügt werden. D.h.,

OpenStack und andere Cloud OS unterscheiden nicht zwischen CPU und Core. Diese unklare Begrifflichkeit bedarf besonderer Beachtung, andernfalls führt sie zur Verwirrung.
Hinzu kommt bei jeder VM eine virtuelle Peripherie bestehend aus Festplatten, Netzwerkkarten und anderen Benutzerendgeräten. VMs laufen in voneinander getrennten Adressräumen im virtuellen Speicher des Host OS und sind sowohl voneinander als auch vom Host OS isoliert.
Das Problem bei VMs ist, dass weder die Server-Hardware noch deren Peripherie dafür entworfen wurden, dass mehrere Betriebssysteme auf demselben Rechner ausgeführt werden und simultan auf essentielle Komponenten wie Hauptspeicher und Peripherie zugreifen wollen. Erst seit ca. 2005 gibt es dazu im Server-Bereich hardware-basierte Virtualisierungshilfen in mehr oder weniger fortgeschrittener Ausprägung. Die Implementierung einer virtuellen Maschine in einem Host OS bzw. einem Hypervisor stellt allerdings nach wie vor eine Herausforderung dar. Besonders schwierig ist die Bereitstellung von virtuellem Speichers in einer VM, denn bereits der Speicher der realen Maschine ist virtuell, so dass der virtuelle Speicher einer VM eine doppelte Virtualisierung darstellt. Dem Benutzer ist diese Komplexität nicht bewusst, vielmehr erwartet er von einer VM, dass sie dieselbe Funktionalität bzgl. Speichergröße, Verschnitt und Speicherschutz bietet, wie ein realer Rechner. Leider stellt dieser Wunsch aus folgenden Gründen ein Problem dar:

1. Zwar hat jedes Guest OS eine Verwaltung des eigenen virtuellen Speichers in Form von Guest OS Page Tables. Diese enthalten jedoch sinnlose Hauptspeicheradressen, da jeder Zugriff auf realen Speicher vom Hypervisor abgefangen werden muss, genauso wie jeder Zugriff auf die reale MMU des Servers. Würde dies der Hypervisor nicht tun, würde das Guest OS die Kontrolle über die reale Maschine erlangen, und die verschiedenen VMs auf einem Server würden sich gegenseitig stören.

2. Zudem werden vom Guest OS falsche Page Fault Exceptions erzeugt, die ebenfalls der Hypervisor abfangen muss. Die Page Fault Exceptions sind deswegen falsch, weil das Guest OS keine Kenntnis über den tatsächlichen virtuellen Speicher des Host OS und dessen Aufteilung auf Blöcke im Hauptspeicher und auf Records im Swap Device hat.

Um die beschriebenen Schwierigkeiten bei der Realisierung virtuellen Speichers in einer VM besser verstehen zu können, wird zuerst kurz beschrieben, wie normalerweise die Verwaltung von virtuellem Speicher in einer realen Maschine abläuft und danach, wie der Speicher in einer VM virtualisiert wird.

5.7 Virtueller Speicher in einer realen Maschine

In realen Rechnersystemen, die nicht für den Einsatz in Echtzeitanwendungen oder für eingebettete Systeme vorgesehen sind, wird seit langem schon virtueller Speicher verwendet, um über eine Datei, dem sog. swap file, oder eine Partition auf der Festplatte, der sog. swap partition, den real installierten Speicher durch Seitentauschen (paging) scheinbar zu vergrößern.
Weitere Gründe für virtuellen Speicher sind, dass damit der Verschnitt beim Laden mehrerer Programme in den Hauptspeicher wegfällt, und dass damit Programme ausgeführt werden können, die zu groß für den realen Speicher sind, und dass auf diese Weise mehr Programme ausgeführt werden können als gleichzeitig in den realen Speicher passen, und dass damit Programme besser voreinander geschützt werden können. Dazu wird vom Prozessor eine virtuelle Speicheradresse ausgegeben, die mit Hilfe einer Speicherverwaltungseinheit (Memory Management Unit, MMU) in eine physikalische Adresse umgewandelt wird, die die Caches und der Hauptspeicher verstehen. Die Abbildungsfunktion „virtuelle Adresse (virtual address, VA) -> reale Adresse (physical address, PA)“ wird in Teilen in der MMU, und dort in einem schnellen Cache, dem sog. Translation Lookaside Buffer (TLB), als Kopie gespeichert. Da der Fall eines Cache Miss im TLB auftreten kann, wird parallel zum TLB eine Speicherung der kompletten Abbildungsfunktion in den sog. Seitentabellen (Page Tables) im Hauptspeicher abgelegt, die eine Datenstruktur des Betriebssystems darstellen. Im TLB sind aus Kapazitätsgründen nur die am dringendsten und die am häufigsten benötigten Teile der Abbildungsfunktion enthalten, während die Page Tables die Funktion als Ganzes enthalten. Die Verwaltung der Page Tables und des TLB wird vom jeweiligen Host OS übernommen. Allerdings ist der Zugriff auf die Seitentabellen viel langsamer als auf den TLB, so dass nur bei einem TLB Miss von den Seitentabellen Gebrauch gemacht wird.
Um den drastischen Geschwindigkeitseinbruch im Falle eines TLB Miss abzumildern, haben die meisten CPUs, die nicht embedded sind, einen sog. Hardware Page Table Walker (HPTW), der die Page Tables viel schneller durchlaufen und nach einem Eintrag suchen kann, als dies der CPU in Software möglich wäre. Sobald der HPTW im Falle eines TLB Miss den gesuchten Abbildungseintrag in den Page Tables gefunden hat, kopiert er diesen in den TLB, weil dieser Eintrag wahrscheinlich auch in Zukunft gebraucht werden wird. Zeigt die Abbildungsfunktion im so aktualisierten TLBs auf eine Seite, die momentan auf das Swap Device ausgelagert ist, wird dies als Seitenfehlen (page fault) bezeichnet. In diesem Falle tauscht das Host OS die im Hauptspeicher fehlende Seite mit einer Seite, die wenig benötigt wird (page swapping), indem die letztere auf das Swap Device ausgelagert und die erstere an der Stelle der letztere im Hauptspeicher eingelagert wird. Anschließend kann die CPU auf die neue Seite zugreifen, und deren wichtigste und am häufigsten referenzierte Inhalte werden automatisch in den Cache gestellt.

Im Falle eines Seitenfehlens im Hauptspeiche, erzeugt die MMU der CPU eine Page Fault Exception, die das Betriebssystem auf den Plan ruft, das die Seite tauscht. Der Vorgang des Seitentauschens erfordert auch eine Aktualisierung der Statusbits in den Seitentabellen und im TLB und kostet aufgrund der Festplattenzugriffe bis ca. 107 CPU-Takte an Zeit, ist also sehr aufwendig. Durch das preemptive Prozess-Scheduling des Host OS wird diese Latenz dem Benutzer gegenüber verdeckt, indem der Prozess, der eine Page Fault Exception ausgelöst hat, aus der Liste der rechenbereiten Prozesse entfernt und an seiner Stelle ein anderer Prozess bearbeitet wird, der rechenbereit ist. Währenddessen holt das Host OS per DMA die fehlende Seite vom Swap Device und beginnt danach dessen Ende, den Prozess, der de-scheduled war, wieder abzuarbeiten (Re-Scheduling).

5.8 Virtueller Speicher einer virtuellen Maschine

Der virtuelle Speicher in einer virtuellen Maschine ist eine doppelte Illusion. Man muss für diesen Speicher unterscheiden zwischen VM-virtuellen Adressen (Guest Virtual Addresses, VM-VA), VM-realen Adressen (Guest Physical Addresses, VM-PA), virtuellen Adressen des Host OS (Host Virtual Addresses, VA) und realen Adressen des Servers (Host Physical Addresses, PA). VM-virtuelle Adressen (VM-VAs) sind die Adressen, mit der das Guest OS und die Anwendungen arbeiten. Beispielsweise erhält jeder Prozess, der von einem Guest OS verwaltet wird, seinen eigenen virtuellen Adressraum innerhalb des Guest OS. VM-reale Adressen (VM-PAs) sind die Adressen, von denen das Guest OS und seine Prozesse glauben, dass sie real seien. Da jedoch jede VM vom Host OS als Prozess verwaltet wird, sind VM-PA nur virtuelle Adressen im Host OS. D.h., es gibt einen Bereich virtueller Adressen im virtuellen Adressraum des Host OS, der identisch ist mit den VM-PAs einer virtuellen Maschine ist. Außerdem sind die virtuellen Adressen des Servers (VA) diejenigen Adressen, mit denen das Host OS und dessen Prozesse arbeiten. Die realen Adressen des Servers (PA) schließlich sind diejenigen Adressen, die am Adressbus der CPU anliegen und die die Caches und der Hauptspeicher verstehen. Die PAs sind die einzigen Adressen, die tatsächlich in der Hardware existieren. Alles andere ist Software, die durch Hardware unterstützt wird.
Zusammen mit den vier Adresstypen VM-VAs, VM-PAs, VAs und PAs existieren auch verschiedene Abbildungen zwischen diesen, die in Bild 1 dargestellt sind. Die Abbildung VM-VAs -> VM-PAs basiert auf einem Satz von Seitentabellen im jeweiligen Guest OS. Die Abbildung VA -> PA wird von einem zweiten Satz von Page Tables vorgenommen, der dem Host OS gehört. Die Abbildung VM-PAs -> VAs kann entfallen, da alle VM-PAs zugleich VA-Adressen desjenigen Host OS-Prozesses sind, der die VM implementiert. Bei der Virtualisierung von Ressourcen geht es letztlich also nur um die Abbildung VM-VAs -> PAs.

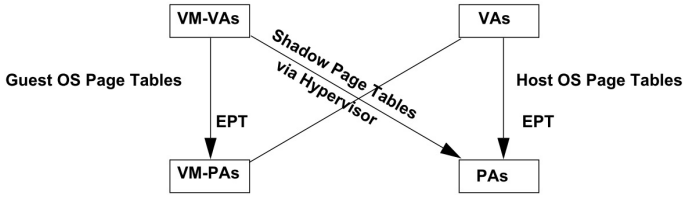

Bild 1: Abbildungen zwischen den vier Adresstypen VM-VAs, VM-PAs, VAs und PAs.

Für diese Abbildung (VM-VAs -> PAs) gibt es zwei Konzepte: Das erste Konzept ist eine Software-basierte Lösung durch den Hypervisor, das zweite verwendet eine Hardware-basierte Virtualisierungshilfe namens Extended Page Tables (EPT) in der CPU und im Hauptspeicher. Die EPTs bilden zuerst VM-VAs auf VM-PAs und danach VM-PAs = VAs auf PAs ab, d.h. es ist ein Abbildungsschritt mehr als üblich erforderlich. Dies bedeutet einen erhöhten Aufwand für die Abbildung, da sowohl durch die Speicherverwaltung des Guest OS als auch die des Host OS gegangen werden muss. Bei dem Hypervisor-Konzept hingegen werden mit Hilfe von sog. Schattenseitentabellen (Shadow Page Tables) im Hypervisor direkt VM-VAs in PAs umgewandelt. Beide Konzepte werden nachfolgend genauer beschrieben.

5.8.1 Software-basierte Adressabbildung für eine VM

Bei der Software-basierten Adressabbildung kopiert der Hypervisor die Guest OS- und die Host OS-Seitentabellen, die eine VM betreffen, in eine interne Datenstruktur, sobald die VM hochgefahren ist. Diese Informationen werden vom Hypervisor benützt, um daraus eine Direktabbildungstabelle VM-VAs -> VM-PAs für jede VM zu erstellen, die sog. Schattenseitentabellen. Die Informationen aus den Seitentabellen des Host OS braucht der Hypervisor deshalb, weil nur diese die realen Verhältnisse im Hauptspeicher widerspiegeln. Mit seinen Schattenseitentabellen kann dann der Hypervisor die Abbildung VM-VAs -> VM-PAs einer VM selber durchführen. Im Laufe der Zeit entstehen so im TLB des Cores, der den Hypervisor ausführt, die wichtigsten Kopien der VM-VA-nach-VM-PA-Abbildung, was diese erheblich beschleunigt.
Zusätzlich „missbraucht“ der Hypervisor das Steuerregister CR3 (CR3 = Control Register 3) in der IA32- und IA32/64-Architektur. Das CR3 existiert in jedem Core einer CPU dieser Architektur und zeigt auf den Beginn einer Seitentabelle, genauer auf die physikalische Adresse des Seitentabellenverzeichnisses. Mit dieser Information im CR3 kann der HPTW des Cores die Seitentabellen durchlaufen, sobald ein TLB Miss im Core auftritt.
Der Hypervisor setzt das CR3 auf seine eigenen Schattenseitentabellen, um

diese bei einem TLB Miss vom HPTW durchlaufen zu lassen. Dazu ist es erforderlich, dass der Hypervisor jeden Versuch des Guest OS, das CR3 auf die Guest OS-Seitentabellen zu setzen, abfängt. Das Abfangen erfolgt über einen Trap. Im Trap wird dann das CR3 auf die Schattenseitentabellen gesetzt. Ein zusätzlicher Vorteil des Abfangens besteht darin, dass das Guest OS nie Zugriff auf einen Hauptspeicherbereich erhält, der außerhalb des für den Guest OS vorgesehen Bereichs von VAs liegt. D.h., der Hypervisor hat stets die Speicherzugriffe aller Guest OS unter seiner Kontrolle.
Im besten Fall (TLB Hit) wird so der TLB nur ein Mal pro Adressabbildung durchlaufen. Im schlechtesten Fall (TLB Miss) muss der Hypervisor seine Schattenseitentabellen vom HWPT durchlaufen lassen und anschließend den TLB aktualisieren. Außerdem muss der Hypervisor auch dann eingreifen, wenn das Guest OS die eigenen Seitentabellen aktualisiert. Dann kopiert er diese Informationen in seine Schattenseitentabellen, um daraus eine aktuelle Direktabbildungstabelle VM-VAs -> VM-PAs zu erstellen, um später diese Abbildung selber zu machen. Für das Guest OS läuft das Geschehen so ab, also ob es keine Virtualisierung gäbe.
Aufgrund des ständigen Kopierens in die Schattenseitentabellen und des Aktualisierens derselben ist diese Art von Adressabbildung ineffizient, wenn das Guest OS oder die Anwendung häufige Änderungen der Guest OS- Seitentabellen aufweist. Da hilft es auch nicht, dass zur Auflösung eines TLB Miss nur eine einzige Datenstruktur, nämlich die Schattenseitentabelle, per HPTW durchlaufen werden muss.

5.8.2 Hardware-basierte Adressabbildung für eine VM

Bei der Hardware-basierten Adressabbildung, wie es z.B. beim EPT-Konzept realisiert wurde, müssen im Falle eines TLB Miss zwei Sätze von Seitentabellen, eine im Guest OS und eine im Host OS durchlaufen werden. Dafür ist der Hypervisor nur minimal involviert, was Zeit spart. Allerdings kommt beim Durchlaufen der Guest OS- und Host OS-Seitentabellen erschwerend hinzu, dass diese i.d.R. mehrfach kaskadiert sind: Ausgehend von einer Basisseitentabelle, die nicht ge-paged, d.h. nicht auf ein Swap Device ausgelagert wird, sind mehrere Zugriffe auf eine Seitentabelle nötig, um eine einzige Adressauflösung durchzuführen. Für eine einzige 64 Bit-VA-Adresse eines Intel IA32/64-Prozessors beispielsweise sind gemäß [11] vier Zugriffe des Host OS auf eine Seitentabelle erforderlich, um daraus eine PA zu erstellen. Aus der Sicht eines Guest OS bedarf es ebenfalls vier Seitentabellenzugriffe, um aus einer VM-VA eine VM-PA zu berechnen. Allerdings erfordert jeder Guest OS-Zugriff auf Guest OS-Seitentabellen eine Adressabbildung VM-VA -> PA bestehend aus vier Host OS-Zugriffen. Hinzu kommt noch ganz zu Beginn eine weitere Adressabbildung VM-VA -> PA mit vier Host OS-Zugriffen, um den Wert des weiter unten beschriebenen Steuerregisters „Guest CR3“ in eine PA des Host OS umzusetzen. Bei der Hardware-basierten Adressabbildung

vervielfacht sich also die Zahl der Seitentabellenzugriffe im Vergleich zur Software-Lösung, was die Behebung eines TLB Miss zeitaufwendig macht. Deshalb ist diese Art der Adressabbildung ineffizient, wenn viele Abbildungen mit TLB Misses ausgeführt werden müssen. Unter dem Strich gibt es keinen klaren Gewinner zwischen Software- und Hardware-basierter Adressabbildung., was die Performanz anbetrifft. Es hängt von den Umständen ab, was besser ist.

5.9 Virtuelle Peripherie einer virtuellen Maschine

Unter der virtuellen Peripherie einer virtuellen Maschine versteht man die Virtualisierung der I/O. I/O-Virtualisierung ist die Fähigkeit eines realen Geräts, gleichzeitig von mehreren VMs genutzt werden zu können. Damit das möglich ist, müssen drei Aufgaben gelöst werden:

1. Es muss ermöglicht werden, dass mehrere VMs im selben Rechner dieselbe reale Peripherie des Rechners im Zeitscheibenverfahren benutzen können. Dies deshalb ein Problem, weil die Peripherie von Haus aus keinen Multiplexbetrieb kennt.

2. Es muss ermöglicht werden, dass die virtuelle Peripherie einer VM über DMA auf den virtuellen Speicher der VM zugreifen kann, um dort ihre Eingabedaten abzulegen bzw. von dort ihre Ausgabedaten zu erhalten. Dies ist deshalb ein Problem, weil dazu das Guest OS den realen DMA-Controller mit der Start- und der Zieladresse des zu transferierenden DMA-Blocks initialisieren müsste. Der VM-Speicher ist jedoch nicht real. Selbst die VM-PAs sind nur ein Bereich in den VAs des Host OS. Außerdem kann das Guest OS den DMA Controller schon deshalb nicht initialisieren, weil ihm dazu die Kernel Mode-Privilegien fehlen.

3. Es muss ermöglicht werden, dass die virtuelle Peripherie einer VM über Interrupts mit einem Gerätetreiber des Guest OS kommunizieren kann. Das ist jedoch aus drei Gründen schwierig: zum einen geht der Gerätetreiber des Guest OS davon aus, dass er exklusiv ein reales Gerät betreibt. In Wahrheit bedient aber der Gerätetreiber des Host OS das Gerät. Zum zweiten kennt der VM-Gerätetreiber nicht die realen I/O-Adressen des Geräts, so dass der Treiber auch nicht dessen Register ansprechen kann. Das Guest OS glaubt jedoch, dass es reale I/O-Adressen ausgibt und mit realer Peripherie arbeitet, was, außer bei der Intel VT-d- Technologie, nie der Fall ist. Zum dritten kann der Gerätetreiber des Guest OS nicht im notwendigen Kernel Mode arbeiten, weil sonst das Guest OS die Kontrolle über die reale Maschine erlangen würde. Ohne den Kernel Mode kann in der IA32-, IA32/64-Architektur kein I/O-Befehl ausgeführt werden. Der Grund dafür ist, dass nur der Kernel Mode in

Ring 0 im Intel-Programmiermodell ausgeführt wird und die Benutzeranwendung und das Guest OS in Ring 3 laufen, mit entsprechend niedrigeren Privilegien.

Durch die I/O-Virtualisierung werden alle drei Aufgaben mehr oder weniger effizient gelöst. Für die Implementierung von I/O-Virtualisierung gibt es, genau wie bei der Speicherverwaltung, eine reine Software-basierte Lösung, die ausschließlich auf dem Hypervisor beruht und wenig effizient ist, sowie eine Lösung aus Hypervisor mit Hardware-Virtualisierungshilfen, die deutliche Effizienzsteigerungen mit sich bringt.

5.9.1 Software-basierte I/O-Virtualisierung

Jeder Gerätetreiber der IA32-, IA32/64-Architektur verwendet für den Gerätezugriff spezielle I/O-Befehle, wie z.B. IORead, IOWrite oder IOStatus, die im Betriebssystem im priviligierten Modus laufen und einen eigenen I/O- Adressraum aufspannen, der getrennt von den virtuellen und realen Hauptspeicheradressen des Betriebssystems ist. Dieser Vorgang wird auch als Ported I/O bezeichnet. Immer dann, wenn der Gerätetreiber des Guest OS auf ein virtuelles Gerät zugreifen will, versucht das Guest OS, auf den Kernel Mode umzuschalten und priviligierte Befehle auszuführen. Dieser Versuch ruft eine Exception (Trap) vom Typ „Access Privilege Violation“ hervor, die der Hypervisor abfängt. Danach kann er folgende Aktionen ausführen:

1. er übersetzt die Guest OS I/O-Adresse in eine Host OS I/O-Adresse, die real ist
2. er übersetzt im Falle eines DMA-Transfers die VM-PAs in PAs des realen Servers
3. er sorgt dafür, dass bei einem DMA-Transfer der gewünschte Hauptspeicherblock tatsächlich im Hauptspeicher eingelagert ist
4. er führt den I/O-Befehl selber aus, diesmal im priviligierten Modus, und gibt das Ergebnis an den Guest OS-Gerätetreiber weiter. Im Falle eines lesenden Zugriffs des Guest OS sind dies die gelesenen Daten und der Gerätestatus. Im Falle eines schreibenden Zugriffs sind es die zu schreibenden Daten und das Gerätekommando, die er an das Gerät weiterreicht.
5. Im Falle eines Interrupts von einem realen peripheren Gerät, verändert er den Interruptvektor, den das Gerät ausgibt, so, dass nicht der Guest OS-Gerätetreiber aufgerufen wird, sondern er selber. Außerdem legt er über den virtuellen Interrupt-Controller der VM einen Interruptvektor an den virtuellen Datenbus der virtuellen CPU der VM an und wickelt deren Interrupt-Protokoll ab, so dass der jeweils richtige Guest OS-Gerätetreiber aufgerufen wird. Der Guest OS-Gerätetreiber reagiert mit

priviligierten I/O-Befehlen auf den Interrupt, die wieder der Hypervisor abfängt und selber ausführt.

6. er nimmt selber das Multiplexen und Demultiplexen der Zugriffswünsche und der Daten vor. Beispielsweise verteilt der Hypervisor für den Fall, dass es sich um eine Netzwerkkarte handelt, die von der NIC empfangenen Datenrahmen auf die Ziel-VMs. Umgekehrt sammelt er zu sendende Datenrahmen von den Quell-VMs ein und schickt die Rahmen nacheinander über dieselbe NIC ab. Zum Multiplexen und Demultiplexen bildet er einen Switch in Software nach, den sog. vSwitch, der bei n VMs n+1 Ports hat. Ein Port ist virtuell mit dem realen I/O-Gerät verbunden, die übrigen n mit den n Guest OS des Host OS.

7. Für den Fall, dass der Guest OS-Gerätetreiber nur generisch und somit gar nicht in der Lage ist, das reale Gerät tatsächlich im Detail zu bedienen, interpretiert der Hypervisor die I/O-Befehle, die der Guest OS-Gerätetreiber ausgibt, und erzeugt eine Folge von Befehlen, die tatsächlich funktionieren. Generische Gerätetreiber kommen im Guest OS z.B. dann vor, wenn eine Cloud Middleware installiert wurde.

Alle Aktionen erfolgen per Software, haben teilweise ein hohe Komplexität und kosten viel CPU-Leistung und I/O-Bandbreite. Messungen von Intel haben gezeigt, dass beispielsweise bei einer 10 Gbit/s-Ethernet-NIC auf diese Weise nur 4,5-6,5 Gbit/s erreicht an Durchsatz werden können. Dies entspricht einem Overhead durch die Virtualisierung von bis zu 55%.

5.9.2 Hardware-basierte I/O-Virtualisierung

Die Hardware-basierte Virtualisierungsunterstützung benutzt entweder die Tatsache, dass moderne Intel/AMD-CPUs eine eigene I/O MMU und verschiedene Verbesserungen integriert haben, oder sie benutzt eine Erweiterung des PCIe-Standards, den sog. Address Translation Service (ATS), oder sie beruht alternativ auf Hardware-Erweiterungen in der Peripherie selber. Für den Fall, dass es sich um bei der Peripherie um eine Graphikkarte handelt, gibt es auch die Variante, dass eine sog. Graphics Address Remapping Table (GART) benutzt wird. Es gibt schließlich auch den Fall, dass sich PCIe- und Grafikadressen in eine virtuelle Maschine „hineinreichen“ lassen. So arbeitet Intel VT-d. Die VM übernimmt bei VT-d die exklusive Kontrolle über ein reales I/O-Gerät. Damit der direkte Zugriff einer virtuellen Maschine auf die Grafikkarte bzw. das I/O-Gerät korrekt abläuft, müssen allerdings auch das BIOS und das Mainboard die I/O-Virtualisierung unterstützen.
Um die Hardware-basierte I/O-Virtualisierung in einem späteren Kapitel näher erläutern zu können, werden an dieser Stelle die Informationen zu PCIe gegeben, die notwendig sind.

5.10 PCIe-Konzepte für die Virtualisierung

PCI Express (PCIe) ist im Gegensatz zu PCI kein Bus, sondern ein leitungsvermitteltes, Punkt-zu-Punkt Übertragungsmedium, das mehr Datendurchsatz als ein Bus erzielt, bei gleichzeitig geringerer Latenz. Ein PCI Express-basierter Rechner besteht aus der Sicht von PCIe aus drei Teilen: 1.) dem sog. Root Complex, 2.) den PCI Express-Endgeräten und 3.) deren I/O-Funktionen, die wiederum auf dem sog. PCIe Configuration Space beruhen.

5.10.1 Der Root Complex

Der Root Complex ist die Southbridge von Intel, die heutzutage auch unter dem Namen I/O Controller Hub (ICH) bekannt ist. Die Southbridge ist für den Anschluss von nicht sehr schnellen PCI Express-Endgeräten an die CPU zuständig. Die schnelle PCI-basierte Graphikkarte und der Hauptspeicher werden mittlerweile direkt mit der CPU verbunden. Früher war dazu die Northbridge notwendig, die diese Komponenten untereinander und mit der Southbridge verband. Die Southbridge/ICH stellt für jedes anzuschließende Endgerät einen bitseriellen PCI Express-Anschluss zur Verfügung. Der Datentransfer zwischen dem Hauptspeicher und einem Endgerät wird über einen PCIe Switch realisiert, der Teil der Southbridge ist.

5.10.2 PCI Express-Endgeräte

Alle Geräte, die eine PCIe-Schnittstelle haben, wie z.B. eine Ethernet-NIC sind PCI Express-Endgeräte. In jedem Endgerät gibt es 4096 Byte an PCIe-Konfigurationsregistern, den sog. Configuration Space.

5.10.3 I/O-Funktionen von PCI Express-Endgeräten

Die I/O-Funktionen eines PCIe-Endgeräts ist die Menge aller Kommandos, die das Endgerät ausführen kann. Dazu zählen das Senden und Empfangen von Daten, die Gerätekonfiguration und die Statusabfrage des Geräts.

5.10.4 PCIe Address Translation Service (ATS)

Virtuelle I/O-Adressen entstehen immer dann, wenn der Gerätetreiber eines Guest OS auf die reale Peripherie eines Servers zugreifen will, z.B. um das Gerät zu initialisieren, zu steuern, oder um einen Datenblock per DMA und Interrupt zwischen Gerät und VM-Hauptspeicher zu transferieren. Für die Adressierung von realen Geräten durch eine VM kann der PCIe Address Translation Service (ATS) [8] verwendet werden, der von der PCI SIG erstmalig im Jahre 2007 spezifiziert wurde. Im Falle von DMA arbeitet der PCIe ATS zusammen mit dem TLB der Server MMU, da virtuelle VM-Adressen (VM

VAs) in reale Hauptspeicheradressen (PAs) umgewandelt werden müssen. Mit Hilfe von PCIe ATS ist außerdem Speicherschutz zwischen PCIe-Geräten möglich.
Gemäß der PCIe ATS-Spezifikation gibt es einen Translation Agent (TA), eine Address Translation Table (ATPT), sowie einen oder mehrere Address Translation Caches (ATCs). Der Translation Agent ist eine spezielle MMU nur für I/O, die in der CPU angesiedelt ist (früher in der Northbridge). Die Address Translation Table ist das Gegenstück zu den Seitentabellen einer konventionellen MMU und ist in einem Speicherbereich untergebracht, der über PCIe erreicht werden kann. Der Address Translation Cache schließlich entspricht dem TLB einer konventionellen MMU. Das besondere bei ATS ist, dass ein Address Translation Cache in jedem PCIe-Gerät existieren kann, wodurch er gerätespezifisch wird und vom Translation Agent getrennt ist. Damit ATC und TA zusammenarbeiten können, wurden in der PCIe ATS-Spezifikation spezielle Transaktionen definiert, die über den Root Complex laufen, der als Bindeglied zwischen TA und PCIe-Gerät fungiert.

6 Intel Virtualization Technology for x86- Architectures VT-x

Intel VT-x besteht aus der Virtual Machine Extension (**VMX**) im Befehlssatz der CPU, der Virtual Maschine Control Structure (**VMCS**) im Hauptspeicher und den Extended Page Tables (**EPT**) in der CPU und im Hauptspeicher. Alle drei Technologien werden nachfolgend beschrieben.

6.1 Virtual Machine Extension (VMX)

Die Virtual Machine Extension (VMX), die Intel im Jahre 2005 im Befehlssatz eingeführt hat, waren ein großer Schritt nach vorne. Man kann sagen, dass ohne VMX und die sog. VMCS die Virtualisierung eines Rechners deutlich langsamer und unsicherer war. Die im Rahmen von VMX neu eingeführten Befehle sind gemäß [9] VMLAUNCH, VMRESUME, VMXON, VMXOFF, sowie VMPTRLD, VMPTRST, VMCLEAR, VMREAD und VMWRITE. Damit können Hypervisoren VMs mit höherer Effizienz starten, fortsetzen, verwalten und beenden als nur mit Software, die auf den üblichen IA32-Befehlen beruht. Es gibt weiterhin zwei Arten von VMX-Befehlen: solche mit Root-Privilegien, die im Kernel Mode laufen, und solche mit Non-Root-Privilegien. Interessanterweise gibt es kein von aussen lesbares Status-Bit in der CPU, das die beiden Modi unterscheidet, so dass das Guest OS daran nicht erkennen kann, dass es nur ein Gastbetriebssystem ist. Root-VMX-Befehle sind dem Hypervisor bzw. dem Host OS vorbehalten, Non-Root-VMX-Befehle können auch von den Guest OS ausgeführt werden. Der Übergang von Root zu Non-Root

heißt „VM entry“, die Rückkehr von Non-Root zu Root ist der VM exit. Die VMX Befehle bedeuten im einzelnen:

- VMXON: der VMX-Befehlssatz wird aktiviert. Dieser Befehl wird zu Beginn vom Hypervisor durchgeführt.
- VMXOFF: der VMX-Befehlssatz wird deaktiviert. Dieser Befehl wird beim Herunterfahren des Hypervisors durchgeführt.
- VMLAUNCH: der Hypervisor startet eine neue VM. Dadurch wird auch ein VM entry-Übergang in die neue VM vollzogen.
- VMRESUME: der Hypervisor startet eine VM wieder, die bereits existierte und als Binärfile, dem sog. „Image“, abgespeichert war.
- VMWRITE, VMREAD und VMCLEAR: das sind Befehle, mit denen der Hypervisor die weiter unten beschriebene VMCS-Datenstruktur initialisiert, schreibt und liest.
- VMPTRST (= VM Pointer Set), VMPTRLD (= VM Pointer Load): Damit wird der Pointer, der auf die VMCS-Datenstruktur zeigt, geschrieben und gelesen.

Die VMX-Befehle arbeiten eng mit der Virtual Maschine Control Structure (VMCS) zusammen, die im nächsten Kapitel beschrieben wird.

6.2 Virtual Maschine Control Structure (VMCS)

Die Non-Root-VMX-Befehle eines Guest OS und die Übergänge VM entry und VM exit werden durch die Virtual Maschine Control Structures (VMCS) [10] beeinflusst, die Datenstrukturen im Hauptspeicher darstellen. Für jeden virtuellen Core in einer virtuellen CPU wird vom Hypervisor eine VMCS angelegt, die den Status des Cores widerspiegelt. Im virtuellen Hauptspeicher einer VM existieren so viele VMCS wie die VM an virtuellen Cores hat. Auf die VMCS wird über einen durch VMX verwalteten Pointer zugegriffen. Eine VMCS ist komplex strukturiert und weist verschiedene Datenfelder für den Status von Guest OS und Host OS auf, die bei einem VM entry-Übergang gelesen und bei VM exit geschrieben werden. Ein Datenfeld in der VMCS, die VM execution control, steuert die Non-Root-VMX-Befehle, die der Core ausführt. Außerdem gibt es zwei weitere Felder, VM entry control und VM exit control, die die gleichnamigen Übergänge steuern. Das letzte Feld in der VMCS, die VM exit information, speichert den Grund, warum die VM verlassen und zum Hypervisor gesprungen wurde. Die Semantik jedes VMCS Datenfeldes ist komplex und aufwendig und spiegelt letztendlich die Schwierigkeiten wider, die die Virtualisierung mit sich bringt. Deshalb soll hier nur die Grundidee der Aufgaben von VMCS aufgezeigt werden:

Zum Abfangen einer Root-Instruktion, die von einem Core in einer VM abgesetzt wird, benutzt der Hypervisor die VM entry/VM exit-Übergänge. Bei der anschließenden Emulation der Root-Instruktion durch den Hypervisor muss sichergestellt werden, dass dabei nicht aus Versehen der Hypervisor-Zustand oder der Zustand einer VM sich verändern. Hierzu wird die VMCS-Datenstruktur benutzt. Sie speichert den Zustand der Register des Cores in der Guest State Area der VMCS ab. Darüberhinaus speichert die VMCS den Registerzustand desjenigen Cores, der den Hypervisor ausführt, in der Host State Area ab. Mit den Informationen aus Guest State und Host State Area kann der Hypervisor jederzeit den ursprünglichen Zustand von sich selbst und des virtuellen Cores auf dem Server wiederherstellen.

6.3 Intel Extended Page Tables (EPT) bzw. AMD RVI

Seit der Nehalem-Architektur gibt es bei Intel die Extended Page Tables (EPT) [11], [12] Sie bewirken zusammen mit dem Hypervisor, dass jede VM die Illusion eines eigenen virtuellen Hauptspeichers erhält, der zudem einigermaßen effizient verwaltet wird. EPT erlaubt jeder VM, eigene Seitentabellen (Page Tables) im Guest OS zu etablieren und diese zu modifizieren, ohne, dass dazu der Hypervisor eingreifen muss. EPT beruht im Falle eines TLB Miss auf der Kaskadierung der Page Tables von Host OS und Guest OS, mit der Konsequenz, dass die Adressumsetzung von virtuellen VM-Hauptspeicheradressen (VM-VAs) auf physikalische Hauptspeicheradressen (PAs) viel Zeit benötigt. Ein HPTW durchläuft dabei die Guest OS- und die Host OS Page Tables. Pro realem Core existiert der HPTW einmal, genau wie es pro Core je einen TLB für den 1st Level-Datencache, den 1st Level-Befehlscache und den 2nd Level Cache gibt. Der HPTW durchsucht die Seitentabellen aufgrund der Speicherhierarchie im Rechner zuerst im 1st Level- Datencache, bevor er im Falle eines Cache Miss auf den 2nd und den 3rd Level Cache der CPU und danach auf den Hauptspeicher und schließlich auf die Festplatte zugreift.
Da von einem realen Core mehrere VMs im Time-Sharing-Betrieb abgearbeitet werden können, können auch mehrfach hintereinander TLB Misses auftreten, die von verschiedenen VMs herrühren. Diese müssen von demselben HPTW nacheinander aufgelöst werden. Dadurch und aufgrund der Tatsache, dass die Page Table Walks des HPTW kaskadiert sind, gibt es bei EPT Leistungsverluste, die allerdings i.A. geringer ausfallen als bei der reinen Software-Lösung. Die EPT sind ineffizient, wenn viele Abbildungen mit TLB Misses ausgeführt werden müssen. Um den Zeitaufwand bei einem TLB Miss zu reduzieren, existieren bei EPT zwei CR3-Steuerregister, eines für das Guest OS und eines für das Host OS, damit der einzige HPTW desjenigen Cores, der das Host OS ausführt, sowohl die Seitentabellen vom Guest OS als auch vom Host OS durchlaufen kann. Zudem werden die Seitentabellenzugriffe so

ineinandergesteckt, als ob es nur eine einzige Seitentabelle gäbe, die vielfach kaskadiert ist. Dies wird auch als Nested Page Tables bezeichnet. Der entsprechende Vorgang heißt auch Secondary Address Translation (SLAT). Durch das Ineinanderstecken unterschiedlicher Seitentabellen durchläuft der HPTW diese in einem Schema, das gemäß [11] in Bild 2 dargestellt ist. Die Schattenseitentabellen des Hypervisors werden bei EPT nicht mehr benötigt, womit sie auch nicht mehr aktualisiert werden müssen, was eine erhebliche Zeitersparnis bedeutet. Dadurch ist die Hardware-basierte Adressabbildung dann effizienter als die Software-basierte Abbildung, wenn nicht zu viele TLB Misses auftreten.

6.3.1 Leistungsgewinn durch EPT/RVI

Die Fa. VMware beziffert in [13] den Gewinn durch EPT/RVI auf bis zu 48% für allgemeine Benchmarks und auf bis zu 600% für Micro-Benchmarks spezieller Anwendungen, die MMU-intensiv sind. Damit lohnen sich EPTs, allerdings kann bei zu vielen TLB Misses auch ein Leistungsverlust durch EPT auftreten.

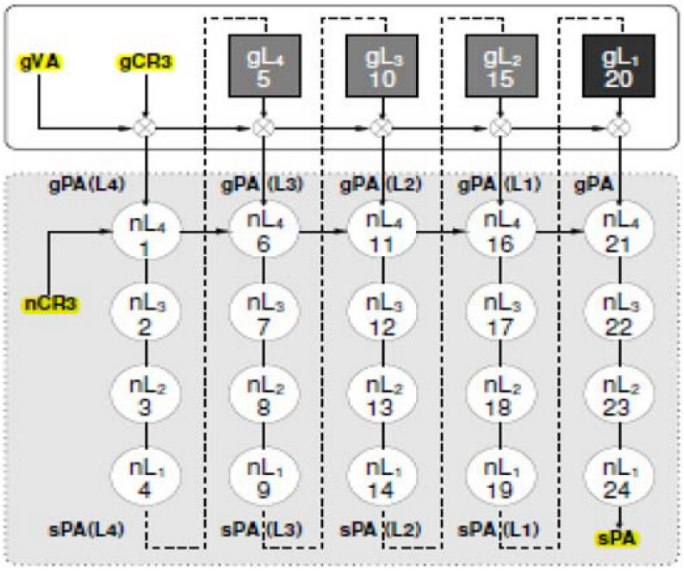

Bild 2: zweidimensionales Schema von Seitentabellen, das der HPTW gemäß [11] durchläuft. gVA = Guest Virtual Address = VM-VA; sPA = System Physical Address = PA; gCR3 = Guest OS CR3 und nCR3 = Host OS CR3.

6.3.2 Leistungsverlust durch EPT/RVI

Die Größe des TLBs eines Cores ändert sich nicht durch das Starten von neuen VMs, die er ausführen soll. Die Zahl der dringenden und wichtigen Adressumsetzungen nimmt aber zu, je mehr VMs auf einer realen Maschine ausgeführt werden. Mit der Zahl von VMs pro Core steigt auch die Zahl ihrer TLB Misses. Von einer kritischen Anzahl von TLB Misses an kommt es zu Leistungseinbußen durch EPT/RVI, da dann der HPTW die Page Tables tief unten in der Speicherhierarchie, d.h. bei der Festplatte durchlaufen muss. Im Fall, dass ein Seitentauschen erforderlich ist, kostet ein TLB Miss ca. 107 CPU Takte pro Festplattenzugriff.
Um die Zahl der TLB Misses zu reduzieren, kann man die Seitengröße in einer Page Table erhöhen, da dann - bei konstanter Hauptspeichergröße- die Gesamtzahl der Seiten sinkt. Je kleiner die Seitenzahl ist, desto kleiner ist die Wahrscheinlichkeit, dass eine bestimmte Seite ausgelagert ist, und desto kleiner ist die Wahrscheinlichkeit für einen TLB Miss. Es wird deshalb von Herstellern empfohlen, die Seitengröße des Host OS von z.B. 4 KB auf bis zu 2 MB zu erhöhen, falls der Server eine hohe Zahl von TLB Misses zu verzeichnen hat. Allerdings steigt mit größerer Seitengröße auch die Transferzeit für das Seitentauschen an, und damit auch die Reaktionszeit des Rechnersystems, was z.B. für Echtzeit ungünstig ist.

6.3.3 Unterstützung von Intel EPT / AMD RVI durch Hardware und BIOS

Die EPT benötigen einen erweiterten HPTW, der mit kaskadierten Page Tables, wie in Bild 2 dargestellt, umgehen kann. Daher ist es ratsam, die Dokumention der CPU daraufhin zu überprüfen, ob die CPU einen solchen HPTW hat. Wenn der HPTW EPT unterstützt, dann muss dieses im BIOS explizit aktiviert werden. Falls danach viele TLB Misses auftreten, sollte es wieder abgeschaltet werden.

6.3.4 Unterstützung von Intel EPT / AMD RVI durch Hypervisoren

Von Xen wird sowohl EPT als RVI unterstützt. Beides ist bereits ab Version 3.4.0 in der Standardkonfiguration aktiviert. Dies kann anhand des Xen-Konfigurationseintrags „hap=1“ überprüft werden. Auch von KVM wird EPT/RVI unterstützt, und ist in der Standardkonfiguration aktiviert, was unter /sys/module/kvm_intel/parameters/ept überprüft werden kann. Für den effizienten Betrieb einer Cloud ist es erforderlich, dass diese Parameter gesetzt sind.

7 Intel Virtualization Technology for Connectivity VT-c

Die Intel Virtualization Technology for Connectivity (VT-c) besteht derzeit aus der I/O Acceleration Technology (**I/OAT**), der Single bzw. Multi Root I/O Virtualization (**SR-IOV**, **MR-IOV**), den Virtual Machine Device Queues (**VMDq**) und der Virtualization Technology for Direct I/O (**VT-d**).

7.1 Intel I/O Acceleration Technology (I/OAT)

Die Intel I/O Acceleration Technology (I/OAT) [14], [15] wurde ab dem Jahre 2006 schrittweise eingeführt. Sie ist keine Virtualisierungshilfe im eigentlichen Sinne, sondern beschleunigt den ganzen Verarbeitungsstrang, der in einem Rechner beteiligten realen Ein-/Ausgabekomponenten. Diese Komponenten sind die PCIe-basierten Endgeräte, wie z.B. Netzwerkkarten, der Intel Northbridge/Southbridge Chipsatz im Falle älterer CPUs, bzw. die Ivy/Sandy Bridge bei neueren CPUs, sowie die CPU selber und den TCP/IP-Protokollstack. Die insgesamt höhere I/O-Effizienz durch I/OAT kommt auch der virtualisierten Hardware zu Gute, weshalb I/OAT hier beschrieben wird.
Darüberhinaus ist Intel I/OAT wichtig für alle Echtzeitanwendungen, die von Server-Systemen ausgeführt werden sollen, da es einige Funktionen enthält, die sich direkt auf eine deterministische und insgesamt niedrige Latenz auswirken. Der Stand bei Intel I/OAT ist der, dass praktisch alle Intel PCIe NICs und CPUs I/OAT-fähig sind.
I/OAT beinhaltet in der CPU die sog. „Quick Data Technology“ und einen „Direct Cache Access“ als Verbesserungen. Die Netzwerkkarten beinhalten eine größere Zahl von Optimierungen, die nachfolgend aufgelistet und beschrieben werden.

- Receive Side Scaling (RSS)
- Quick Data Technology
- Direct Cache Access
- Receive Side Coalescing
- Extended Message-Signaled Interrupts (MSI-X)
- Low Latency Interrupts
- Header Splitting
- TCP Checksum Offload and Segmentation Offload

In diese Liste sind nur diese Technologien aufgeführt, die durch MS Windows unterstützt werden.

7.1.1 Quick Data Technology

Die Quick Data Technology bewirkt, dass das Umkopieren von TCP/IP-Paketen zwischen dem Betriebssystempuffer und dem Datenbereich der Anwendung durch die CPU entfällt. Die Voraussetzung dafür ist, dass das Betriebssystem dies unterstützt, wie es z.B. ab „WindowsServer 2003“ der Fall ist. Ab der Version WindowsServer2008 erfolgt automatisch diese Unterstützung durch eine Features namens NetDMA, ohne, dass dazu etwas konfiguriert werden müsste.

7.1.2 Receive Side Scaling (RSS)

Ab Datenraten von 10 Gbit/s erweist sich ein einzelner Core oft als zu langsam, um die Pakete einer TCP-Verbindung schritthaltend mit ihrem Eintreffen verarbeiten zu können. Receive Side Scaling (RSS) verteilt deshalb eintreffende TCP/IP-Pakete automatisch auf mehrere CPUs innerhalb eines Servers und auf mehrere Cores in jeder CPU, damit diese die Pakete gleichzeitig bearbeiten können. Dadurch findet ein dynamischer Lastausgleich zwischen den Prozessoren statt. Zur Implementierung von RSS gibt es auf jeder NIC mehrere Empfangspuffer in Hardware, wobei jeweils ein Puffer einem Core zugeordnet ist. Welches Paket in welchen Empfangspuffer eingestellt wird, entscheidet eine Hash-Funkti- on, die ebenfalls in Hardware realisiert ist und die als Input den Paket-Header hat. Dadurch entsteht eine pseudozufällige Gleichverteilung von Paketen auf Puffer, um die Puffer möglichst gleichmäßig zu füllen. Da durch das Verteilen der Pakete auf verschiedene Puffer die Paketreihenfolge einer TCP-Verbindung aufgelöst werden könnte, sorgt die Hash-Funktion auch dafür, dass das Auslesen der Pakete aus den Puffern und ihre anschließende Bearbeitung in den Cores so erfolgt, dass die TCP-Sequenznummern in der Reihenfolge eingehalten werden. D.h., Pakete einer TCP-Verbindung, die früher eintrafen, werden auch früher wieder aus ihrem Puffer ausgelesen und bearbeitet. Eine Umordnung von Paketen aufgrund von multiplen Empfangs-Puffern ist im TCP/IP Stack nicht notwendig.

7.1.3 Direct Cache Access (DCA)

Das Ziel von DCA ist es, TCP/IP-Pakete vorauseilend, d.h. kurz bevor sie gebraucht werden, in die Datencaches jedes Cores zu stellen, so dass ein Core-Zugriff auf den langsamen Hauptspeicher nicht nötig wird. Dazu teilt der DCA, der in der NIC angesiedelt ist, einem Core mit, welches Paket demnächst bearbeitet werden soll und wo es im Hauptspeicher steht. Sobald das Paket von der NIC per DMA in den Betriebssystempuffer im Hauptspeicher kopiert wurde, findet in diesem Speicherbereich ein Cache Line Prefetch statt durch den 1st-Level Datencache des Cores statt.

7.1.4 Receive Side Coalescing (RSC)

Beim Receive Side Coalescing (RSC) werden mehrere kleinere TCP/IP-Pakete, die zur selben TCP-Verbindung gehören, in ein großes Paket umgepackt, wodurch die Zahl der Header und Trailer sinkt, was sich günstig auf die Bearbeitungszeit auswirkt. Außerdem reduzieren sich dadurch auch die Zahl der DMA-Transfers und Interrupts. Wenn RSC durch die Software unterstützt wird, d.h. durch Guest und Host OS, dann ist damit ein erheblicher Performance-Gewinn durch die beschriebene Blockbildung möglich, die allerdings auf Kosten der Latenz geht.

7.1.5 Extended Message-Signaled Interrupts (MSI-X)

Die Extended Message-Signaled Interrupts (MSI-X) erlaubt es jedem Empfangspuffer, der von RSS verwendet wird, dass der Puffer selbst und nicht die NIC, in der er untergebracht ist, den ihm zugeordneten Core unterbrechen kann, sobald der Puffer ein Paket zur weiteren Bearbeitung eingespeichert hat. Durch MSI-X erhält jeder Empfangspuffer einen eigenen Interrupt-Vektor, so dass gleichzeitig mehrere Interrupt-Service Routinen von genauso vielen Cores bearbeitet werden können, wie es RSS-Empfangspuffer gibt.

7.1.6 Low Latency Interrupts (LLI)

Die Low Latency Interrupts (LLI) sind wichtig für Echtzeitdatenverarbeitung, denn damit kann man solche Prozesse priorisieren, die bei der Datenübertragung empfindlich auf hohe oder auf schwankende Latenzen reagieren. LLI ist ein Software-Werkzeug, dass es erlaubt, darin Latenz-sensitive Prozesse einzutragen. Mit diesen Angaben reguliert LLI automatisch die Frequenz, mit der ein Core, der für einen Latenz-sensitiven Anwenderprozess zuständig ist, unterbrochen wird, um diesen Prozess periodisch zu bearbeiten. Der Core kann im Time-Sharing auch andere Prozesse bearbeiten, die nicht zeitkritisch sind. Diese werden in der Bearbeitung verzögert, während der zeitkritische Prozess öfters als die anderen ausgeführt wird und immer dann, wenn der Core per Interrupt unterbrochen wurde.

7.1.7 Header Splitting

Durch Header Splitting wird es ermöglicht, dass bereits nach dem Eintreffen eines Paket-Headers im Daten-Cache eines Cores die Bearbeitung durch eine Interrupt Service Routine beginnen kann, ohne, dass dazu der Rest des Pakets, bestehend aus Nutzlast und CRC, im Cache eingetroffen sein muss. Dies verkürzt die Paketlatenz. Die Technik ist verwandt mit dem bekannteren virtual cut-through Verfahren bei Routern.

7.1.8 TCP Checksum Offload und Segmentation Offload

Beim TCP Checksum Offload wird die Berechnung der TCP-Prüfsumme auf die NIC verlagert und dort in Hardware ausgeführt. Dies funktioniert auch für UDP- und IP-Pakete. Beim Segmentation Offload wird das Aufteilen einer Socket-Nachricht in einzelne TCP-Pakete auf die NIC ausgelagert. Beides sorgt für Beschleunigung des I/O.

7.2 Virtual Machine Device Queues (VMDq)

Die Virtual Machine Device Queues (VMDq) wurden im Jahre 2008 eingeführt und stellen im Vergleich zu SR-IOV eine veraltete Virtualisierungshilfe dar. Dementsprechend ist VMDq auch langsamer als SR-IOV, da es noch eine Software-basierte Unterstützung durch den Hypervisor beim Umkopieren von Puffern erfordert, die es bei SR-IOV nicht mehr gibt. Technisch gesehen ist VMDq ein Hardware-Beschleuniger auf einer PCIe-Netzwerkkarte, der einige, aber nicht alle Aufgaben des Hypervisors beim Zugriff auf das LAN oder das Internet abnimmt. Gemäß [16] gibt es bei einer NIC mit VMDq-Unterstützung auf der Netzwerkkarte einen sog. Rahmen-Sortierer und Klassifizierer, der auf der OSI-Schicht 2 arbeitet. Darüberhinaus existieren auf der NIC so viele Hardware-Puffer zum Senden und Empfangen, wie es virtuelle Maschinen gibt, die auf die NIC zugreifen wollen. Bei einigen NICs ist allerdings von Haus aus die Zahl der Puffer auf n=8 begrenzt.
Die Aufgabe von VMDq besteht in der Zuordnung von Datenrahmen zu Sende- und Empfangspuffern. Sobald ein Datenrahmen vom Internet in der NIC eintrifft, legt der Sortierer und Klassifizierer den Rahmen in einen von n Empfangspuffern ab. Die Einspeicherung erfolgt anhand der Zieladresse des Rahmens, die eine MAC-Adresse ist, und anhand einer etwaigen VLAN-Adresse im Rahmen. Allerdings ist VMDq nicht in der Lage, zwischen zwei Rahmen zu unterscheiden, die zwar dieselbe MAC-Zieladresse haben, aber unterschiedliche VLAN-Adressen. Sie werden in denselben Puffer eingestellt. Umgekehrt wird vom Sortierer und Klassifizierer ein zu sendender Rahmen in eine der n Sendepuffer auf der NIC eingespeichert. Die Zuordnung beim Senden erfolgt anhand der Herkunfts-MAC-Adresse des Rahmens, d.h. anhand der VM, die die Nutzlast für den Senderahmen erzeugt hat. Die n Sendepuffer werden vom NIC-Steuergerät auf der Karte gemäß Round-Robin Scheduling bedient, um so eine Fairness beim Senden zu erreichen.
Die beschriebenen Vorgänge sind ähnlich zu denen, wie sie bei SR-IOV ablaufen. Parallel dazu arbeitet allerdings beim VMDq der vSwitch des Hypervisors in Software daran, die 2n Sende-/Empfangspuffer der realen Netzwerkkarte und die n virtuellen NICs der VMs zu bedienen, indem er permanent zwischen diesen Instanzen Datenrahmen hin- und herkopiert. Sobald ein Datenrahmen kopiert wurde, wird von dem betreffenden Puffer auf der NIC ein sog. MS-X Interrupt ausgelöst, der wiederum Teil von Intel I/OAT ist.

Der Gewinn von VMDq besteht darin, dass der Hypervisor nicht mehr schritthaltend mit dem Datentransfer Rahmen entgegenehmen muss, denn dies stellt eine Echtzeitaufgabe dar, die den Hypervisor bei 10G NICs schnell überfordern kann. Der zweite Vorteil von VMDq ist, dass der Hypervisor nicht mehr die Zuordnung von Rahmen zu Sende- und Empfangspuffern vornehmen muss.
VMDq kann alleine oder zusammen mit VT-d eingesetzt werden. In Kooperation mit VT-d entfällt das Umkopieren der Puffer zwischen NIC und VMs durch den Hypervisor, was eine deutliche Leistungssteigerung mit sich bringt. VMDq sollte allerdings nicht zusammen mit SR-IOV betrieben werden, da SR-IOV bereits alleine den vSwitch überflüssig macht und eigene Sende- und Empfangspuffer im PCIe-Endgerät für sog. VFs und PFs bereitstellt, und die Datenrahmen per DMA zwischen NIC Puffern und VFs/PFs hin- und her kopiert.
Insgesamt erlauben sowohl VMDq als auch SR-IOV eine erweiterte Nutzung von PCIe-basierten Endgeräten, die i.d.R. Ethernet-NICs sind. Beide Techniken haben als Ziel, den bei der Virtualisierung durch den vSwitch des Hypervisors entstehenden Software-Overhead beim Multiplexen und Demultiplexen von Datenrahmen zu reduzieren und schließen sich dadurch gegenseitig aus.

7.2.1 Unterstützung von VMDq durch einen Hypervisor

7.2.1.1 KVM VMDqs werden von KVM unterstützt, sind dort allerdings unter dem Feature „Multiqueue virtio-net“ bekannt. Informationen wie VMDq in KVM aktiviert werden kann finden sich unter:
http://www.linux-kvm.org/page/Multiqueue#Enable_MQ_feature.

7.2.1.2 VMware ESXi VDMqs werden auch von VMware ESXi 4.0 unterstützt. VMDq heißt dort NetQueue-Technologie.

7.2.2 Unterstützung von VMDq durch die Netzwerkkarte

Informationen, ob eine NIC VDMq unterstützt, finden sich in der jeweiligen NIC-Dokumentation.

7.3 Single Root I/O Virtualization (SR-IOV)

Die ursprüngliche Definition der Single Root I/O Virtualization (SR-IOV) wurde von einer PCI Special Interest Group (PCI SIG) bereits im Jahre 2007 gegeben und wurde seit dem mehrfach aktualisiert [19]. SR-IOV beruht darauf, PCIe-Endgeräte Multi-VM-fähig zu machen, so dass ein solches Gerät reihum mehrere VMs im Multiplexbetrieb bedienen kann. SR-IOV bietet für

jede angeschlossene VM eine eigene, private Bedienschnittstelle, VF oder PF genannt, die über ein Geräte-internes Multiplexing und mit Hilfe eines individuellen Satzes von Bedienregistern pro VM auf dieselbe reale Hardware-Schnittstelle zugreift, die es nur einmal gibt. SR-IOV ist unabhängig von dem konkreten PCIe-Gerät, wird aber meist mit PCIe-NICs eingesetzt. SR-IOV ist auch unabhängig von darüber liegenden Netzwerk-Protokollen wie TCP/IP, sowie von deren Spezialversionen Open-MX und MXM. Die Live-Migration einer VM mit SR-IOV ist im Gegensatz zu VT-d grundsätzlich möglich, befindet sich zur Zeit allerdings noch in einem experimentellen Status.
Hinweis: Siehe dazu einen Konferenzbeitrag von 2013:
http://insidehpc.com/2013/09/virtual-machine-migration-sr-iov-infiniband/

7.3.1 Die Funktionsweise von SR-IOV

SR-IOV stellt eine Erweiterung des PCI Express-Standards dar und erlaubt es einem realen Endgerät, wie mehrere virtuelle Endgeräte zu erscheinen. Jedes virtuelle Endgerät kann statisch einer VM zugeordnet werden, die es exklusiv betreibt. Mit SR-IOV wird das Multiplexen und Demultiplexen von Datenrahmen in einer realen NIC durch den vSwitch des Hypervisors überflüssig, da dies bereits auf der Netzwerkkarte durch SR-IOV erfolgt. Das Prinzip von SR-IOV ist es, hardware-basierte Queues in der NIC zur Verfügung zu stellen, zusammen mit einem Hardware-basierten Round-Robin Time-Sharing der NIC queues. Gemäß [20] hat bei SR-IOV ein einzelnes PCIe-Gerät mehrere sog. PCI Konfigurationsbereiche (configuration spaces), zusammen mit deren Basisadressregistern. Jeder Konfigurationsbereich verfügt über einen eigenen I/O-Adressbereich im PCIe-Adressraum des Root Complex. Darüberhinaus hat das SR-IOV PCIe-Gerät so viele Sende- und Empfangspuffer, wie es PCI Konfigurationsbereiche gibt. Das bedeutet, dass ein SR-IOV-fähiges PCIe-Gerät von sich aus für einen Multiplexbetrieb mit mehreren VMs geeignet ist.
In der SR-IOV-Terminologie wird ein PCIe-Konfigurationsbereich entweder als virtuelle Funktion (VF) oder als physikalische Funktion (PF) bezeichnet. VFs sind vereinfachte Versionen von PFs. Jede VF und jede PF stellen eine Bedienschnittstelle für das Gerät dar, die einer VM zugeordnet werden kann. VFs erlauben das Senden und Empfangen von Daten, ein Umkonfigurieren der PCIe-Geräts ist damit allerdings nicht möglich. Dies bleibt den PFs vorbehalten, die den vollen Funktionsumfang des Geräts beinhalten. Immerhin kann eine VF auch das Gerät virtuell rücksetzen. Betroffen ist davon allerdings nur die VF selber, nicht das Gerät, da dies die anderen, parallel laufenden Schnittstellen des Geräts stören würde. Die SR-IOV-Erweiterungs -spezifikation erlaubt es jedem realen Gerät, gleichzeitig mehr als 256 VFs zu haben. Das ist mehr als in der Hauptspezifikation von PCIe 3.0 im Kapitel „Alternative Routing-ID Interpretation (ARI)“ vorgesehen ist. Im Prinzip könnte beispielsweise eine einzige SR-IOV NIC 1000 virtuelle NICs (vNICs)

nachbilden, die alle gleichzeitig senden und empfangen könnten, was allerdings unrealistisch ist. Tatsächlich können z.B. 10 NICs mit einer Bandbreite von jeweils ca. 1 Gbit/s von einer 10 Gbit/s SR-IOV NIC emuliert werden. Außerdem muss jede virtuelle NIC vom Hypervisor bzw. dem Netzwerk-Dienst einer Cloud mit einer MAC-Adresse ausgestattet werden, um verwendungsfähig zu sein.
Der Hypervisor kann dynamisch eine oder auch mehrere VFs und/oder PFs einer VM zuordnen. Dies erfolgt so, dass der Hypervisor diejenigen I/O - Adressbereiche des PCIe-Geräts auf die virtuellen I/O-Adressen des Guest OS abbildet, die er für geeignet hält. D.h., das Guest OS, sieht das, was der Hypervisor will. In den meisten Fällen wird nur der Adressbereich einer einzigen VF einem Guest OS zugeordnet. Allerdings erhält eine vom Hypervisor speziell ausgewählte VM auch mindestens eine PF des Geräts zugeordnet. In dieser besonderen VM ist der sog. Master-Treiber angesiedelt, der das Gerät konfiguriert, seinen Status abfragt und die Fehlerbehandlung bei einer I/O-Exception macht.
Nach einer Initialisierungsphase durch den Hypervisor kann der Address Translation Service des PCIe-Root Complexes ohne, dass dazu der Hypervisor nötig wäre, die einer VM zugeordneten, realen I/O-Adressbereiche des PCIe-Geräts auf virtuelle I/O-Adressen des Guest OS abbilden und umgekehrt. Der Address Translation Service von PCIe beruht auf der IOMMU des Servers, die heutzutage in der CPU untergebracht ist und früher in der Northbridge angesiedelt war, sowie auf dem Root Complex und auf dezentralen Adressübersetzungs-Caches (ATCs) in den einzelnen PCIe-Endgeräten. Damit kann das Guest OS direkt auf ausgewählte PCI Konfigurationsbereiche des Endgeräts zugreifen. Dies ist vergleichbar mit einer anderen Virtualisierungstechnik namens VT-d, bezieht sich aber im Gegensatz zu VT-d nur auf die jeweilige VF oder PF und nicht auf das ganze Gerät. Darüberhinaus ist bei SR-IOV ein Software Switch (vSwitch) im Hypervisor für das Multiplexen/Demultiplexen von Daten nicht mehr notwendig, da diese Vorgänge bereits im PCIe-Endgerät erfolgen. Ohne SR-IOV sorgt entweder der vSwitch oder die VMDq-Technik für das Multiplexing und Demultiplexing von Datenrahmen mehrerer VMs auf eine reale NIC.

7.3.2 Initialisierung eines SR-IOV-Geräts

Die Initialisierung eines SR-IOV-Geräts erfolgt in mehreren Schritten:

1. Der Hypervisor definiert über eine PF des Geräts n>1 PCIe-Konfigurations-bereiche in den Konfigurationsregistern des Geräts. Jeder Konfigurationsbereich stellt eine VF oder eine PF dar. Zudem wird für jeden Konfigurationsbereich im Speicher des Geräts je ein Schreib- und ein Lese-FIFO bzw. bei NICs je ein Sende- und ein Empfangspuffer alloziert.

2. Der Hypervisor initialisiert den Interruptvektor im Gerät so, dass im Falle eines Interrupts nicht das Host OS sondern er selber aktiviert wird.
3. Falls es sich bei dem Gerät um ein NIC handelt, erfolgt die Vergabe von MAC-Adressen an die VFs oder PFs mit Hilfe derselben oder einer anderen PF der NIC. Die Vergabe wird entweder vom Hypervisor, oder vom Netzwerkservice einer Cloud, oder von dem Master-Treiber vorgenommen.
4. Der Hypervisor initialisiert die I/O MMU so, dass die virtuellen I/O-Adressen eines Guest OS auf die realen I/O-Adressen einer bestimmten VF im Gerät abgebildet werden. Dadurch wird dieses Guest OS der VF zugeordnet. Innerhalb des Guest OS ist wiederum ein VF-Gerätetreiber für die VF zuständig.
5. Der Guest OS VF-Gerätetreiber legt in seinem virtuellen Hauptspeicher aus VM-VA-Adressen einen Sende- und einen Empfangspuffer für nachfolgende DMA-Transfers von und zur VF fest.

7.3.3 Beispielablauf bei SR-IOV

Als Beispiel für den Ablauf bei SR-IOV wird in Bild 3 der Empfang eines Ethernet-Rahmens gemäß [20] schrittweise dargestellt.

1. Ein Ethernet-Rahmen trifft an einer SR-IOV Ethernet-NIC ein
2. Der einlaufende Rahmen wird in der NIC in einem seiner zuvor allozierten Empfangs-FIFOs zwischengespeichert. Dadurch wird der Rahmen einer VF zugeordnet. Welcher Empfangs-FIFO bzw. welche VF das ist, hängt von den MAC-Adressen des Rahmens und der VF ab. Beide müssen zusammenpassen.
3. Vom Gerät wird ein DMA-Transfer gestartet. Dazu werden vom DMA-Controller im Gerät VM-VA-Adressen ausgegeben.
4. Die Adressabbildungs-Hardware bildet die VM-VA-Adressen des Geräts auf PA-Adressen ab, so dass der Rahmen an die richtige Stelle im Host OS abgelegt wird, und zwar dort, wo das Guest OS seinen Speicherbereich hat. Nach dem Ende des DMAs steht der Rahmen im Empfangspuffer des VF-Gerätetreibers des Guest OS.
5. Der DMA-Controller löst mit Hilfe eines Interrupt-Controllers im Gerät einen Interrupt bei der realen CPU aus. Dieser wird vom Hypervisor bearbeitet.
6. Der Hypervisor löst einen virtuellen Interrupt im VF-Gerätetreiber des Guest OS aus, um diesen zu informieren, dass ein Datenrahmen eingetroffen ist.

7. Der VF-Gerätetreiber behandelt den Interrupt, indem er, versucht, priviligierte I/O-Befehle auszuführen. Diese werden vom Hypervisor abgefangen und selber ausgeführt. Das Ergebnis gibt der Hypervisor an den VF-Gerätetreiber zurück. Dadurch bedient der VF-Gerätetreiber des Guest OS indirekt die VF.

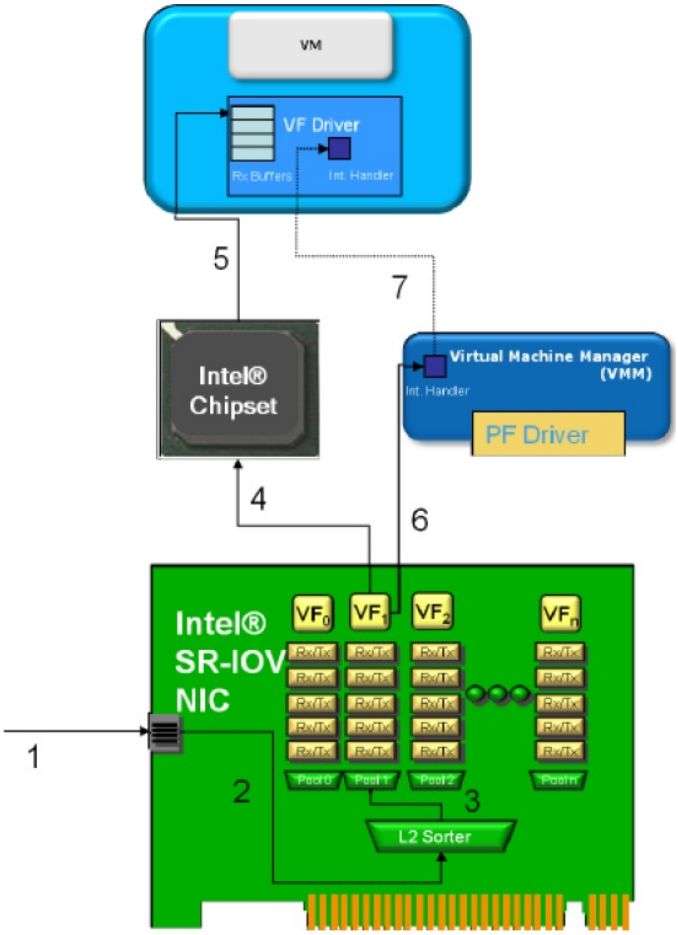

Bild 3: Ablauf beim Empfang eines Datenrahmens bei SR-IOV nach [20]. Der Chipsatz ist z.B. eine Intel/AMD CPU mit IOMMU und separater Southbridge oder die Kombination aus Northbridge und Southbridge.

Der Gewinn gegenüber einer rein Software-basierten I/O-Virtualisierung liegt im DMA-Transfer der Daten und in der Adressübersetzung in Hardware, die beide wesentlich schneller als in Software ablaufen. Besonders bei 10G-NICs

ist dies wichtig, da ein üblicher Core andernfalls nicht schritthaltend mit der Rate eintreffender Daten diese verarbeiten kann.

7.3.4 Multi Root I/O Virtualization (MR-IOV)

SR-IOV erlaubt es, dass ein PCI Express-Endgerät von mehreren VMs, die sich auf der selben physikalischen Maschine befinden, im Multiplexbetrieb gemeinsam genutzt werden kann. Dies gilt allerdings nicht für VMs auf verschiedenen physikalischen Maschinen, die miteinander gekoppelt sind, wie es z.B. bei Blade Server der Fall ist. Diese Lücke füllt MR-IOV, indem es die SR-IOV Spezifikation um Funktionen für Blade Server erweitert. MR-IOV ist genau wie SR-IOV ein PCI Express-Standard.

7.3.5 Voraussetzungen für SR-IOV

Damit SR-IOV funktioniert, müssen diverse Voraussetzungen erfüllt sein. Die wichtigste davon ist, dass der Rechner einen Address Translation Service für PCIe-Adressen haben muss. Dies ist entweder der Fall, wenn das ATS der PCIe SIG benützt wird, oder wenn Intel VT-d eingesetzt wird. Nur dadurch sind SR-IOV-fähige PCIe-Geräte in der Lage, Daten zwischen dem virtuellen Speicher eines Guest OS und dem Gerät direkt zu transferieren. Daneben muss entweder das BIOS oder der Hypervisor diejenigen PCIe-Konfigurationsbereiche anlegen, die für die PFs und VFs notwendig sind. Außerdem muss der Hypervisor zusammen mit einem PF- und einem VF-Treiber die angelegten PCIe-Konfigurationsbereiche den VMs mitteilen, damit sie diese Schnittstellen benützen können. Schließlich muss der Hypervisor den VMs erlauben, ihre VF zurückzusetzen.
Es sollte anhand des BIOS des Motherboards und der Dokumentation der installierten PCI Express-Endgeräte geprüft werden, ob deren PCI Express-Schnittstellen SR-IOV unterstützen. Außerdem muss SR-IOV vom Hypervisor bzw. vom HOST-Betriebssystem, sowie vom GuestOS der VM und von der Cloud Middleware (OpenStack) unterstützt werden. Wann das der Fall ist, wird nachfolgend beschrieben.

7.3.5.1 SR-IOV-Unterstützung durch KVM und Red Hat Enterprise Linux

Hierbei wird KVM als Hypervisor zusammen mit Red Hat Enterprise Linux (RHEL) eingesetzt. Für diesen Fall wird SR-IOV ab RHEL-Version 6.1 unterstützt.

7.3.5.2 SR-IOV-Unterstützung durch XEN

SR-IOV wird von allen neueren XEN-Versionen unterstützt. Da es sich bei Xen um einen Typ-1 Hypervisor handelt, wird Xen ohne HostOS eingesetzt.

7.3.5.3 SR-IOV-Unterstützung durch das GuestOS SR-IOV muss vom GuestOS durch einen erweiterten Gerätetreiber unterstützt werden -den VF-Gerätetreiber- damit das Guest OS auf die VFs zugreifen kann. Erweiterte Gerätetreiber findet man z.B für Intel NICs im RHEL-Distributionspaket „ixgbevf“.

7.3.5.4 SR-IOV-Unterstützung durch OpenStack SR-IOV wird von OpenStack unterstützt. Mehr Informationen sind verfügbar unter:
https://wiki.openstack.org/wiki/PCI_passthrough_SRIOV_support

7.3.6 Zusammenhang zwischen SR-IOV und EPT

SR-IOV kann zusammen mit den Extended Page Tables EPT benutzt werden.

7.3.7 Die Bedeutung von SR-IOV für Rechenzentren

SR-IOV ermöglicht eine bessere Energie- und Kosteneffizienz als eine Softwarebasierte I/O-Virtualisierung. Dies ist darin begründet, dass letztere viel CPU-Rechenleistung kostet, was z.B. dazu führt, dass eine reale Netzwerkkarte nicht mit voller Geschwindigkeit betrieben werden kann, da ein einzelner Core dafür zu langsam ist. Dies liegt u.a. am vSwitch des Hypervisors und an der Software-Adressabbildung von Adressen, die die CPU-Zeit, den Speicherplatz und die Latenz in die Höhe treiben. Durch den Einsatz von SR-IOV findet das Rahmen-Multiplexen/Demultiplexen bereits in der Netzwerkkarte statt, und die Adressabbildung erfolgt per Hardware, was zu einer erheblichen Entlastung des Hypervisors, d.h. der CPU führt. Freiwerdende CPU-Ressourcen können anderweitig verwendet werden, oder der Server kann in den Power Down Modus gehen. Dies erlaubt es, dass die gleiche Rechenleistung mit weniger Servern erreichbar ist und dass einzelne Server aufgrund geringer Auslastung abgeschaltet werden können.
Eine besondere Bedeutung kommt SR-IOV in Kombination mit 10G Ethernet-/Infiniband-NICs oder noch schnelleren NICs zu, da solche Karten oft nur mittels Virtualisierung voll ausgelastet werden können. So ist es mit SR-IOV möglich, 10 VMs mit jeweils 1 Gbit/s über eine einzige reale 10G NIC effektiv anzubinden. Daraus resultiert eine bessere Energiebilanz, da neun reale NICs mit jeweils 1 Gbit/s eingespart werden können. Da eine 10 G SR-IOV NIC nicht wesentlich mehr Energie verbraucht als eine 1G NIC ohne SR-IOV gilt der Satz: Je mehr 1G NICs das Rechenzentrum betrieben hat und durch 10G SR-IOV NICs ersetzt, desto mehr Energie kann es einsparen, d.h. die Ersparnis skaliert mit der Größe des Rechenzentrums.

7.4 Intel VT-d bzw. AMD Vi

Intel VT-d bzw. AMD Vi wurde bereits im Jahre 2006 eingeführt und ist deshalb eine rel. alte Virtualisierungshilfe. VT-d wurde aber im Laufe der Jahre aktualisiert und ist auch heute noch in Verwendung.

7.4.1 Funktionsweise von Intel VT-d

Das Prinzip von Intel VT-d bzw. AMD Vi ist es, dass der Hypervisor Peripheriegeräte einer VM statisch zuordnen kann, und dass der Hypervisor den DMA-Transfer zwischen Guest OS und dem zugeordneten Gerät auf Basis von VM-PA-Adressen organisieren kann, um so die Geschwindigkeit und die Zuverlässigkeit der I/O-Virtualisierung zu erhöhen. Im einzelnen leistet die Virtualization Technology for Directed I/O (Intel VT-d/AMD Vi) folgende Aufgaben:

1. Zuordnung eines realen Peripheriegeräts, meist auf PCIe-Basis, zu einer VM bzw. dessen Guest OS (= I/O Device Assignment)

2. Adressabbildungen für DMA-Transfers zwischen dem realen Peripheriegerät und der VM (= DMA Remapping). Zum einen wird dabei für den DMA-Transfer vom Guest OS zum Gerät der Bereich virtueller Guest OS-Adressen des Sendepuffers in PAs des realen Servers umgewandelt, so dass das Gerät seine Daten von der richtigen Stelle im realen Hauptspeicher holen kann. Zum anderen wird für den DMA vom Gerät zum Guest OS der virtuelle Guest OS-Adressbereich des Empfangspuffers in PAs des realen Servers umgewandelt, so dass das Gerät seine Daten an die richtige Stelle im realen Hauptspeicher ablegen kann. Darüberhinaus werden die virtuellen I/O-Adressen des Guest OS in reale I/O-Adressen des Geräts übersetzt und umgekehrt, so dass das Guest OS die realen Register des Geräts lesen und schreiben kann. Da dazu priviligierte I/O-Befehle notwendig sind, wird vom Guest OS die Hilfe des Hypervisors benötigt. Für die Adressabbildungen werden I/O Page Tables im Hauptspeicher angelegt und verwendet, die unabhängig von den Page Tables der Server MMU sind. Die I/O Page Tables werden von einem I/O TLB ge-cached, der Teil einer I/O MMU ist. Diese war früher in der sog. Northbridge untergebracht, befindet sich aber mittlerweile in der CPU selber.

3. Umleitungen von Interrupts (= Interrupt Remapping). Ein Interrupt des Geräts startet nicht einen Gerätetreiber des Host OS, sondern den Hypervisor, der den Interrupt abfängt und an den passenden Gerätetreiber des Guest OS weiterleitet. Dazu erzeugt der Hypervisor einen virtuellen Interrupt im Guest OS.

4. Bearbeiten von Gerätefehlern (=I/O Exception Handling), um schädliche Auswirkungen auf das Host OS zu verhindern.

VT-d arbeitet bzgl. der Adressabbildung für DMA-Transfers und dem Abfangen von Interrupts ähnlich wie SR-I/O, hat allerdings nicht dessen multiplen PCIe-Konfigurationsbereiche. Damit fallen auch die VF- und PF-Schnittstellen von SR-IOV weg. Die Konsequenz ist, dass ein PCIe-Gerät nur einer einzigen VM zugeordnet werden kann. Mit VT-d wird quasi ein komplettes PCIe-Gerät an die VM „durchgereicht“. Dies wird manchmal auch als PCI Pass Through bezeichnet. So kann beispielsweise auf einem Server, der über eine NIC mit einem Ethernet-Anschluss verfügt, eine VM auf diese Netzwerkkarte zugreifen, sobald das Gerät, der Hypervisor und die VM VT-d unterstützen. Dies bringt erhebliche Einschränkungen bzgl. der Skalierbarkeit des Systems mit sich: das System ist nicht mehr skalierbar. somit kann VT-d bezüglich der Funktionalität als Teilmenge von SR-IOV betrachtet werden.

7.4.2 Zusammenhang zwischen VT-d und SR-IOV

VT-d und SR-IOV sind jeweils für sich alleine lauffähig, können aber auch miteinander kombiniert werden. Die Adressübersetzung von VT-d kann von SR-IOV dazu verwendet werden, um eine VF oder PF eines SR-IOV-fähigen Geräts an eine VM durchzureichen. Die Adressübersetzung von VT-d kann auch n VFs/PFs gleichzeitig an n zugeordnete VMs durchreichen. Für SR-IOV macht es keinen Unterschied, ob VT-d beteiligt ist oder nicht, da es sowieso eine exklusive Nutzung einer VF durch eine VM vorausgesetzt.
Einschränkungen beim Einsatz VT-d entstehen dadurch, dass derzeit bei VMs, die VT-d nutzen, keine Live-Migration der VMs mehr möglich ist. D.h., wenn SR-IOV zusammen mit VT-d eingesetzt wird, ist auch SR-IOV nicht mehr migrationsfähig. Allerdings ist auch die Live-Migration von SR-IOV ohne VT-d bislang nur in einem experimentellen Stadium, so dass dies in der Praxis keine echte Einschränkung darstellt.
Es ist zu erwarten, dass dies mittelfristig ändern wird, da sowohl bei den PCIe-Hardware-Herstellern, als auch den Hypervisor-Entwicklern eine starke Forschungsaktivität in dieser Hinsicht zu verzeichnen ist und eine prototypische Umsetzung einer VT-d-Migration bzw. ihrer VM bereits erfolgreich war. Bis es soweit ist, kann die fehlende Live-Migration von VT-d z.B. durch replizierte VMs, Load Balancing und IP Swapping kompensiert werden.

7.4.3 Zusammenhang zwischen VT-d und EPT

VT-d kann zusammen mit den Extended Page Tables EPT benutzt werden.

7.4.4 Voraussetzungen für VT-d

Damit VT-d innerhalb und außerhalb einer Cloud funktioniert, muss sowohl das Motherboard als auch die PCIe-Karte entsprechend ausgerüstet sein.

7.4.4.1 VT-d-Unterstützung durch das Motherboard Bei Servern mit Intel- oder AMD-CPU kann die I/O MMU entweder auf dem Motherboard in einem Intel/AMD-Chipsatz oder in der CPU implementiert sein. Eine Unterstützung von VT-d muss deshalb entweder von der CPU oder von dem Chipsatz auf dem Motherboard gegeben sein. Diese Unterstützung ist oft nicht aktiviert, sondern muss im BIOS explizit eingeschaltet werden. Wie das erfolgt, ist in [22] beschrieben.

7.4.4.2 VT-d-Unterstützung durch die PCI Express-Peripherie Damit PCIe-Peripheriegeräte über VT-d an VMs durchgereicht werden können, müssen diese VT-d und die PCIe-Funktion „function level reset“ (FLR) unterstützen. Ob das jeweilige Peripheriegerät FLR unterstützt, steht in der Gerätedokumentation.

7.4.4.3 VT-d-Unterstützung durch OpenStack VT-d wird von OpenStack unterstützt. Weitere Informationen finden befinden sich unter:
https://wiki.openstack.org/wiki/Pci_passthrough

8 Spezialversionen von TCP

Bei den Spezialversionen von TCP geht es darum, dass bei TCP nur diejenige Funktionalität installiert wird, die wirklich gebraucht wird. Da HPC- Anwendungen sehr oft die Kommunikationsbibliothek MPI verwenden und MPI oftmals auf TCP aufsitzt, ist es außerdem notwendig, die TCP-Schnittstelle, die sog. Berkeley Sockets, beizubehalten, damit alle HPC-Anwendungen unverändert lauffähig sind.
Dafür gibt es zwei Open Source Lösungen namens „MXM“ [23] und „Open-MX“ [24]. Beide sorgen dafür, dass für Rechner, die im selben Raum oder in derselben Halle aufgestellt sind, diejenigen TCP-Funktionen wegfallen, die nur für die weltweite Datenübertragung nötig sind. Diese Funktionen sind für räumlich konzentrierte Server nicht nützlich und nicht notwendig. Beispiele dieser Funktionen sind der adaptive Time Out, die adaptiven Segmentlängen und die Stauvermeidungskontrolle. Ohne diese Funktionen läuft TCP sehr viel schneller, und MPI-basierte, parallele oder verteilte Programme sind effizienter. Nachfolgend werden sowohl MXM als auch Open-MX beschrieben.

8.1 Mellanox MXM

MXM ist die Messaging Acceleration-Software der Fa. Mellanox, die keine überflüssige TCP-Funktionen enthält und die gleichzeitig die Standard-TCP-Schnittstelle beibehält, so dass z.B. MPI unverändert läuft. MXM ist kostenlos verfügbar, allerdings werden nicht alle Netzwerkkarten (NICs) und Switches unterstützt. Es muss deshalb zuerst geprüft werden, ob die installierten NICs und Switches von MXM unterstützt werden.

8.2 Myrinet Open-MX

Die Alternative zu MXM ist „Myrinet Express over Generic Ethernet Hardware“ (Open-MX) von der Fa. Myrinet. Open-MX ist Open Source Software und sollte laut Beschreibung mit jeder Ethernet-NIC und mit allen modernen Hypervisoren und Linux-Versionen funktionieren. Es leistet dasselbe wie MXM, allerdings ist die Effizienz geringer. Es sollte deshalb vom Systemadministrator eine Entscheidung getroffen werden, ob Open-MX oder MXM installiert wird. Ein weiterer Vorteil von Open-MX ist, dass es gleichzeitig mit beliebigen anderen Protokollen d.h. auch mit TCP und IP einsetzbar ist. Last-but-not least gilt: mit Open-MX kann es TCP auf jedem Rechner zweimal geben, einmal als Vollversion für diejenigen Anwender, die die volle Funktionalität benötigen, und einmal als reduzierte Version für die Nutzung in räumlich konzentrierten Clustern und Clouds.

9 Resümee

Das Konzept der Ressourcen-Virtualisierung von Rechnerkomponenten bis hin zu ganzen virtuellen Rechnern (Virtual Machines, VM) gibt es bereits seit den 1960er Jahren und wurde beispielsweise von IBM in deren Mainframes „System 360“ und „System 370“ sehr erfolgreich eingesetzt. Im Jahre 2005 begann Intel unter dem Namen Virtual Machine Extension (VMX) mit der Erweiterung ihres Befehlssatzes der Pentium-Prozessoren, d.h. der IA32 und IA32/64-Architekturen, um so die Verwendung und das Rechnen auf virtuellen Maschinen (VMs) hardware-mäßig zu unterstützen. Die neu hinzugekommenen Befehle zielten darauf ab, VMs mit möglichst hoher Effizienz zu starten, zu betreiben, zu verwalten und zu beenden. Damit sollte der Verwaltungszusatzaufwand, den Hypervisoren und VMs mit sich bringen, verkleinert werden, was auch gelang. Die Folge der Einführung der neuen Befehle war, dass eine einzelne VM bzgl. der Ausführung eines sequentiellen Programms fast so schnell war, wie die reale Maschine, auf der sie lief. Das war der Beginn der weiten Verbreitung von Hypervisoren und virtuellen Maschinen im Server-Bereich mit Intel und AMD-Prozessoren.

Seit einigen Jahren erweitern Intel, AMD und nahmhafte Hersteller von PCIe-Endgeräten ihre Unterstützung der Virtualisierung, indem sie Hardware-basierte Hilfen auch für die Kommunikation zwischen VMs untereinander und für den Datentransfer zur virtuellen Peripherie hin anbieten. Wenn diese Hilfen richtig eingesetzt werden, wird die virtuelle Kommunikation deutlich effektiver, d.h. ihre Bandbreite nimmt zu und die Latenz geht zurück. Dies ist deswegen möglich, weil wesentliche Aufgaben des Hypervisors auf zusätzliche Hardware verlagert wird, die nicht virtualisiert wird. Dies führt auch zu einer Steigerung der Energieeffizienz, da Cores weniger durch Verwaltungs -zusatzaufwand belastet sind, was wiederum die „Server Consolidation“ erleichtert.
Von einer effektiven Inter-VM-Kommunikation und einem schnellen und deterministischen Datentransfer zwischen VM und Peripherie hängt es außerdem ab, ob Hochleistungsrechnen (High Performance Computing, HPC) und Echtzeit-Anwendungen (Real-Time Computing, RT) effizient auf einer VM oder einer Cloud möglich sind. Beides stellt eine bemerkenswerte Eigenschaft dar, denn weder Hypervisoren noch Clouds wurden ursprünglich für HPC oder Echtzeit entworfen. Mit den im Beitrag dargestellten Techniken ist hohe Effizienz bei Anwendungen, die mittels MPI- oder OpenMP und Infiniband parallelisiert wurden, erreichbar. Ebenso kann die sog. „weiche Echtzeit“ realisiert werden. Weiche Echtzeit heißt, dass es garantiert ist, dass der Mittelwert der Antwortzeit eines Rechnersystems einen vorgegebenen Wert nicht überschreitet. Besitzt ein solcher Rechner, der z.B. als Datenerfassungssystem eingesetzt wird, eine solche Eigenschaft, dann existiert eine obere Schranke für den stochastischen Erwartungswert E der Antwortzeit TResponse dieses Systems.
Mit Hilfe einer weiteren im Beitrag beschriebenen Optimierung im TCP/IP Stack der Host- und Guest-Betriebssysteme einer VM oder einer Cloud, die nur die Berkeley Socket-Schnittstelle beibehält und für das Cluster irrelevante Funktionen entfernt, kann schließlich bei parallelen Anwendungen, die TCP/IP basiert sind, die Effizienz noch einmal deutlich gesteigert werden.

Literaturhinweise

[1] Intel® Ethernet Glossary of Technology Terms, https://intelethernet-hp.com/wp-content/uploads/2011/05/Intel-Glossary-of-Terms_HP_0511_Final.pdf, 2011, abgerufen am 24.07.2014.

[2] http://www.intel.com/content/www/us/en/virtualization/virtualization-technology-capabilities-detail.html, 2014, abgerufen am 24.07.2014.

[3] https://01.org/blogs/tlcounts/2014/virtualization-advances-performance-efficiency-and-data-protection

[4] Werner Fischer, Virtualisierungsfunktion Intel VT-x aktivieren, http://www.thomas-krenn.com/de/wiki/Virtualisierungsfunktion_Intel_VT-x_aktivieren, 2013, abgerufen am 24.07.2014.

[5] H. Richter, A. Obeid, M. Glukhikh, M. Moiseev, Layer 1 and 2 of a Ring-based, Real-Time Network for In-Vehicle Communication, in Proc. of „IEEE 6th International Conference on Ultra Modern Telcommunication and Control Systems (ICUMT 2014), http://www.icumt.info/2014/, St. Petersburg, RU, Oct. 2014.

[6] Intel Telecom and Compute Products, Intel® Virtualization Technology for Connectivity, http://www.slideshare.net/Cameroon45/intel-virtualization-technology-for-connectivity, 2009, abgerufen am 24.07.2014.

[7] PCI Express, PCIe® Base 3.0 Specification, 2014, https://www.pcisig.com/specifications/pciexpress/base3/, abgerufen am 24.07.2014.

[8] PCI Express, Address Translation Service, https://www.pcisig.com/members/downloads/specifications/iov /ats_r1.1_26Jan09.pdf, 2009, abgerufen am 24.07.2014.

[9] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3, http://www.intel.de/content/www/de/de/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.html, 2014, abgerufen am 27.07.2014.

[10] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3, pp. 7, http://www.intel.de/content/www/de/de/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.html, 2014, abgerufen am 27.07.2014.

[11] J. Ahn, S. Jin, J. Huh, Revisiting Hardware-Assisted PageWalks for Virtualized Systems, in Proc. 39th International IEEE Symposium on Computer Architecture (ISCA), pp. 476-487, 2012, http://calab.kaist.ac.kr/ jhuh/papers/ahn_isca2012.pdf, abgerufen am 27.7.2014

[12] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3, pp. 107, http://www.intel.de/content/www/de/de/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.html, 2014, abgerufen am 27.07.2014.

[13] VMware, Performance Evaluation of Intel EPT Hardware Assist, 2009, http://www.vmware.com/resources/techresources/10006, abgerufen am 28.07.2014.

[14] Intel, Integrated Network Acceleration Features of Intel® I/O Acceleration Technology and Microsoft® Windows Server®, 2008, http://www.intel.de/content/www/de/de/I/O/i-o-at-windows-2008-integrated-network-acceleration-features-paper.html?wapkw=i%2fo+acceleration+technology+%28i%2foat%29, abgerufen am 28.07.2014.

[15] Intel, Accelerating High-Speed Networking with Intel® I/O Acceleration Technology, 2006, http://www.intel.de/content/www/de/de/I/O/i-o-acceleration-technology-paper.html?wapkw=i%2fo+acceleration+technology+%28i%2foat%29, abgerufen am 28.07.2014.

[16] Intel LAN Access Division, Intel VMDq Technology, in Notes on Software Design Support for Intel VMDq Technology, 2008, http://www.intel.de/content/www/de/de/search.html?keyword=Virtual+Machine+Device+Queues+%28VMDQ%29, abgerufen am 21.07.2014.

[17] http://www.intel.de/content/www/de/de/virtualization/vmdq-technology-paper.html

[18] http://www.intel.de/content/www/de/de/network-adapters/gigabit-network-adapters/I/O-acceleration-technology-vmdq.html?wapkw=virtual+machine+device+queues+%28vmdq%29

[19] PCI SIG, Single Root I/O Virtualization and Sharing Specification, 2010, http://www.pcisig.com/specifications/iov

[20] Intel LAN Access Division, PCI-SIG SR-IOV Primer - An Introduction to SR-IOV Technology, http://www.intel.com/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf, Januar 2011, abgerufen am 22.07.2014.

[21] Intel, Intel Virtualization Technology for Directed I/O - Architecture Specification, http://www.intel.de/content/www/de/de/intelligent-systems/intel-technology/vt-directed-I/O-spec.html, 2013, abgerufen am 24.07.2014.

[22] Werner Fischer, Virtualisierungsfunktion Intel VT-d aktivieren, http://www.thomas-krenn.com/de/wiki/Virtualisierungsfunktion_Intel_VT-d_aktivieren, abgerufen am 24.07.2014.

[23] Mellanox Technologies, MellanoX Messaging Library User Manual, http://www.mellanox.com/page/products_dyn?product_family=135 &menu_section=73 2014, abgerufen am 24.07.2014.

[24] Inria, Open-MX Myrinet Express over Generic Ethernet Hardware, http://open-mx.gforge.inria.fr/, 2013, abgerufen am 24.07.2014.